

2

Defense Information Systems Agency Technical Integration Support (DISA-TIS)

AD-A262 642



DTIC
ELECTE
S APR 9 1993 D
C



DISTRIBUTION STATEMENT A

Approved for public release
Distribution Unlimited

MUMPS Study

1993

93-07433



| REPORT DOCUMENTATION PAGE | | | Form Approved OMB No. 0704-0188 | |
|--|------------------------|---|------------------------------------|----------------------------|
| <small>Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.</small> | | | | |
| 1. AGENCY USE ONLY (Leave blank) | 2. REPORT DATE 1993 | 3. REPORT TYPE AND DATES COVERED final | | |
| 4. TITLE AND SUBTITLE Defense Information Systems Agency Technical Integration Support (DISA-TIS) MUMPS Study | | 5. FUNDING NUMBERS | | |
| 6. AUTHOR(S) Nancy Hoffer | | | | |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Defense Information Systems Agency (DISA) Center for Information Management (CIM) Office of Technical Integration (OTI) | | 8. PERFORMING ORGANIZATION REPORT NUMBER | | |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) DISA/CIM/OTI 5201 Leesburg Pike, Suite 1501 Falls Church, VA 22041-3201 | | 10. SPONSORING/MONITORING AGENCY REPORT NUMBER | | |
| 11. SUPPLEMENTARY NOTES | | | | |
| 12a. DISTRIBUTION / AVAILABILITY STATEMENT Available to the public | | 12b. DISTRIBUTION CODE | | |
| 13. ABSTRACT (Maximum 200 words) The document details a study of the Massachusetts General Hospital Utility Multi-Programming System (MUMPS) programming language and DoD Automated Information Systems (AISs) written in MUMPS that support the health business area. MUMPS is an interactive programming language and general purpose database management system. It has been primarily used to develop medical information systems. Implementations of MUMPS are available for mainframes, minicomputers, workstations and personal computers. The strength of the language is in its ability to manipulate and store unstructured textual information. Due to its emphasis on textual data, MUMPS does not support mathematical or scientific functions especially well. | | | | |
| 14. SUBJECT TERMS Technical Integration MUMPS Health (CIM) Collection | | Medical Database Management System | | 15. NUMBER OF PAGES 312 |
| 17. SECURITY CLASSIFICATION OF REPORT unclassified | | 18. SECURITY CLASSIFICATION OF THIS PAGE unclassified | | 16. PRICE CODE |
| | | 19. SECURITY CLASSIFICATION OF ABSTRACT unclassified | | 20. LIMITATION OF ABSTRACT |

EXECUTIVE SUMMARY

The Office of Technical Integration requested a study of the Massachusetts General Hospital Utility Multi-Programming System (MUMPS) programming language and Department of Defense (DoD) Automated Information Systems (AISs) written in MUMPS that support the health business area.

Currently 3 of the 26 AISs supported within DoD Health Affairs are based on the MUMPS language. The relevant DoD policy issues regarding MUMPS pertain to its continued use within DoD:

- Can MUMPS be appropriately considered a Higher Order Language as defined in DoDD 3405.1?
- Is MUMPS an Advanced Software Technology (AST) as defined by the Assistant Secretary of Defense (Command, Control, Communication, and Intelligence) in his memo of 17 April 1992?
- Considering recent DoD initiatives, is MUMPS an appropriate language for large AISs and is it well suited to meet Open Systems Environment requirements?

MUMPS, developed in 1967 by the Massachusetts General Hospital Laboratory of Computer Science, is an interactive programming language and general purpose database management system. It has been primarily used to develop medical information systems. Implementations of MUMPS are available for mainframes, minicomputers, workstations, and personal computers. The strength of the language is in its ability to manipulate and store unstructured textual information. Due to its emphasis on textual data, MUMPS does not support mathematical or scientific functions especially well. MUMPS has been approved as an American National Standard (ANS/MDC X11.1-1990), a Federal Information Processing Standard Publication (FIPS PUB 125), and an International Standards Organization/International Electrotechnical Commission Standard (ISO/IEC 11756). The MUMPS Development Committee (MDC) is responsible for maintaining the MUMPS language standard.

MUMPS STUDY REPORTS

The study of the MUMPS language produced the following reports:

- Department of Veterans Affairs (VA) position on and use of the MUMPS programming language.
- Use of the MUMPS programming language within DoD Health Affairs.
- Advanced Software Technology plans for MUMPS FIPS PUB

| | |
|--------------------|-------------------------------------|
| Accession For | |
| NTIS CRA&I | <input checked="" type="checkbox"/> |
| DTIC TAB | <input checked="" type="checkbox"/> |
| Unannounced | <input checked="" type="checkbox"/> |
| Justification | |
| By | |
| Distribution/ | |
| Availability Codes | |
| Dist | Avail and/or Special |

A-1

- Analysis of MUMPS as a Higher Order Language
- Analysis of MUMPS as an Advanced Software Technology

These reports collectively address the relevant language issues, use of the MUMPS language within DoD and other Federal Agencies, and DoD policies towards the use of programming languages, as summarized below.

VA Position and Use of MUMPS

The VA is the largest and most influential user of the MUMPS language. The VA has used MUMPS since 1982, primarily to develop and maintain its Decentralized Hospital Computer Program (DHCP). DHCP has been implemented at 172 VA hospitals and 12 clinics at an estimated cost of \$800 million to \$1 billion. DHCP supports a wide range of hospital functions such as patient registration, clinical applications, and administrative functions. The VA has found the strengths of MUMPS to be an excellent fit for the functional requirements of DHCP. The VA is committed to using MUMPS for DHCP maintenance and development for at least the next 10 years. Based on its experience the VA considers MUMPS to be superior to other languages in productivity, maintainability, ease of learning, and portability. The VA provides a central role in the continued development of the MUMPS language. A large portion of the proposals for changes to the language standard adopted by the MDC are sponsored by the VA.

DoD Health Affairs use of MUMPS

Health Affairs is the primary user of the MUMPS language within the DoD. Health Affairs uses MUMPS as the primary language for the Expense Assignment System III (EAS III), the Automated Quality of Care Evaluation Support System (AQCESS), and the Composite Health Care System (CHCS). EAS III is used to provide consistent uniform reporting of expenses and manpower at 200 military medical treatment facilities (MTFs). AQCESS is used as a quality assurance system at roughly 160 MTFs. CHCS is a hospital information system (HIS) that will eventually replace AQCESS and other systems in use at MTFs (patient scheduling, appointments, pharmacy, radiology, patient administration, and laboratory). The DoD Major Automated Information Systems Review Committee has approved deployment of portions of CHCS to all military hospitals over a 7 year period of time. Health Affairs has identified several improvements that could be made in the MUMPS language:

- Improved support for data integration and data interchange
- Improved support of an Open Systems Environment (OSE)
- Additional performance tuning tools for MUMPS databases

AST Plans for MUMPS FIPS PUB

A new MUMPS FIPS PUB is scheduled to be released shortly. The current FIPS PUB is based on the 1984 ANS. The updated FIPS PUB will reflect the 1990 ANS. The ANS is also scheduled to be updated and published in late 1993. The new ANS will lead to future changes in the MUMPS FIPS PUB. Planned improvements in later ANS and FIPS PUBS include block structuring in code, external routine library support, standardized error handling, parameter passing, and improved support for DoD and industry standards. Until these changes are adopted and included, the MUMPS FIPS PUB is deficient in supporting DoD standards and integrating with internal and external routines.

MUMPS as a Higher Order Language

It is the policy of the DoD to use the following products, in order of preference, for AIS development and maintenance:

- off-the-shelf tools and advanced software technology
- Ada-based software and tools
- Approved Higher Order Languages (HOLs)

Ada, C, COBOL, FORTRAN, and Pascal are among DoD's approved Higher Order Languages. Several criteria have been specified by the National Institute of Technology (NIST) and its predecessor, the National Bureau of Standards, for considering a language as a HOL. Overall, MUMPS appears to meet the minimum criteria for a HOL. As HOLs, Ada and C are clearly superior to MUMPS. COBOL, FORTRAN, and Pascal also rate somewhat higher than MUMPS. MUMPS is a suitable alternative for AISs well matched with the strengths of the language. MUMPS satisfies the criteria best when used for small to moderate sized applications that deal primarily with unstructured, textual data. MUMPS is not an ideal alternative for large scale development projects, AISs requiring tight integration with existing non-MUMPS based systems, and AISs requiring comprehensive support for DoD and industry standards.

MUMPS as Advanced Software Technology

The DoD has categorized an Advanced Software Technology as either a Software Tool, Life-Cycle Support Environment, Programming Support Environment, Non-Procedural Language, Database Management System, or Other Technology. An "Other Technology" includes those technologies not expressly enumerated in the definition. In the context of this definition, MUMPS could be considered an Other Technology that integrates a programming language, database management system, and rudimentary programming support environment. The additional overhead required to integrate system components must be weighed against MUMPS's specific weaknesses with respect to a particular component. For small to moderate sized applications dealing with textual data, MUMPS is an appropriate language. MUMPS is less desirable for larger applications or those dependent upon mathematical and statistical

functions. Since the DoD appears to intend for an AST to be suitable for a wide range of application sizes and capabilities, MUMPS appears not to meet the criteria for an AST.

THE CORPORATE INFORMATION MANAGEMENT INITIATIVE

An important consideration in selecting a programming language is determining the extent to which the language is compatible with its intended development environment. The DoD has implemented Corporate Information Management (CIM) to integrate business process improvement and cross functional integration in conjunction with the deployment and use of information resources within DoD. Within the CIM program, economic analyses are performed to justify expenditures in information technology. In considering MUMPS as a language for large AISs, several CIM-related aspects of the language should be considered for its long-term use in DoD.

Interoperability with non-MUMPS systems

Establishing interoperability between AISs is an important objective of the CIM program. Support for external files and subject databases is useful for achieving interoperability and data integration between AISs developed in different programming environments. Although MUMPS possesses strong string parsing capabilities that can be used to build data import and export functions, the weakness of its support for external files and databases increases the difficulty of integrating AISs developed in MUMPS with those developed in other languages.

Portability of program code

The establishment of a portability standard separate from the MUMPS language standard has led to a high degree of portability among MUMPS AISs. Adherence to the portability standard ensures independence between hardware and software, an important objective of the CIM program. The decision to upgrade to a new hardware platform is frequently driven by the cost of modifying application software to operate on the new hardware. By following the guidelines in the MUMPS portability standard the application software will not be the limiting factor in new hardware acquisitions.

Modularity and Reusability

MUMPS forces modularity in program code by explicitly limiting routine size to 4,000 bytes, resulting in programs consisting of many small routines. Constructing larger programs from many smaller routines is considered a good programming practice. However, in certain instances it may be desirable to have a routine that exceeds 4,000 bytes. Recent enhancements to the language let programmers limit the scope of variables, minimizing the potential conflicts with similarly-named variables in other routines. It is more practical to construct a large program from many reusable component parts than to build each component as a custom product. One of the objectives of DoD is to establish a central repository of

these common software components. Like many languages, MUMPS has capabilities to support reusability, although not at the same level as Ada. Ada supports the concept of software reuse exceptionally well through libraries of related routines (packages) and generic routines (generic units). A function can be made sufficiently generic so that the operations it performs can be used as a template for many programs. The internals of these routines are not accessible by the program code. MUMPS currently does not have general standardized libraries of routines. Similar capabilities are often added by implementors in their products.

Maintainability

MUMPS provides a great deal of flexibility to programmers during development because it does not require declarative statements and data types. Furthermore, the language is syntactically very terse, allowing functions to be written with fewer lines of MUMPS code than equivalent functions in other languages. These characteristics of the language may improve programmer productivity during initial development, but these benefits may be outweighed by the adverse impact on program reliability and maintainability, especially in large systems. Although some of these disadvantages can be overcome by establishing and enforcing conventions, MUMPS's emphasis on development productivity at the expense of reliability and maintainability is not consistent with the goals of CIM.

Support for DoD and Open Systems Environment Standards

The DoD Technical Reference Model (TRM), based on the NIST Application Portability Profile, defines a profile of standards and guidelines applicable to all DoD information systems. Although improving, MUMPS currently falls behind other languages in conforming to the standards in the TRM. If MUMPS continues to lack adequate bindings to these standards, integration with other AISs, data repositories, and software repositories within DoD will be hampered. The major goal of the CIM program is to achieve integration by providing a framework of standards for systems analysis and design, software development, and technical architectures. To be usable in DoD, MUMPS must continue to improve in its support of DoD and OSE standards such as SQL, X-Windows, POSIX, PHIGS, etc.

MUMPS and large AISs

Generally, MUMPS is not the ideal language for a very large AIS due to its more limited functionality beyond text manipulation and text data management. Nonetheless, MUMPS has been successfully used to develop certain very large AISs, most notably DHCP. The VA's success with DHCP is due to a variety of factors:

- The VA has been the main influencer of the language and is heavily involved in its standardization.
- The VA is as deeply committed to using MUMPS as DoD is to Ada.

- The VA has the commitment, people, and resources to insure the future success of DHCP.
- The VA does not use MUMPS for purposes for which it was not intended, e.g., statistical and mathematical functions.

Health Affairs is using MUMPS as the language for its large AIS, the CHCS. A modified version of DHCP was used as the base for CHCS, making MUMPS initially the language of choice for this AIS. When developing a very large AIS from scratch, MUMPS becomes much less appealing as a language. C, with its extensive libraries, bindings to standards, and wide range of functions, is overall a more capable language for very large AISs. Similarly, Ada, which has features designed to support large scale development efforts and commitment from DoD, would also provide a better environment over the course of the life-cycle.

Programming Support Environment

The MUMPS programming support environment (PSE) is based primarily on the typical implementation of MUMPS as an interpreter, rather than a compiler. Thus, the programmer can interact directly with the source code during execution and can execute source statements directly at the terminal. The typical MUMPS PSE does not, however, provide configuration management or other features needed to support "programming-in-the-large" (i.e., using a large team of programmers). Nor is it expected that DoD's Integrated Computer-Aided Software Engineering (I-CASE) procurement will support development in MUMPS. In contrast, the standardized Ada Programming Support Environment (APSE) will provide features needed for both programming-in-the large and "programming-in-the-small" (the work of an individual programmer) and will be supported by I-CASE.

Life Cycle Costs

A critical issue to determining the suitability of MUMPS is a comparison of the life cycle costs for building, maintaining, and operating an AIS developed in MUMPS to the costs for using another language. One study (Thomas Munnecke, "A Linguistic Comparison of MUMPS and COBOL", National Computer Conference, 1980) showed that a particular application can be developed and operated in MUMPS using 6% of the lines of code as a comparable version in COBOL. Another study (Dimir M. Alonzo, "System Performance: A Benchmark Study of MUMPS and Other Systems", MUG Quarterly, Vol XIII, 1984) illustrated that MUMPS requires fewer lines of code and provides improved database performance and efficient storage over conventional languages. However, these studies did not focus on maintenance costs. More importantly, the results might well be different for different applications. If DoD truly wishes to consider MUMPS an HOL or AST, it should attempt to estimate the life cycle costs for those applications, such as the next generation CHCS, that it is likely to consider developing in MUMPS.

DEPARTMENT OF VETERANS AFFAIRS POSITION ON MUMPS

TABLE OF CONTENTS

| | Page |
|---|------|
| LIST OF EXHIBITS | iii |
| EXECUTIVE SUMMARY | ES-1 |
| 1.0 INTRODUCTION | 1-1 |
| 1.1 Study Background | 1-1 |
| 1.2 Data Gathering | 1-1 |
| 1.3 Document Organization | 1-2 |
| 2.0 MUMPS BACKGROUND | 2-1 |
| 2.1 Distinctive Characteristics of MUMPS | 2-1 |
| 2.2 MUMPS Standardization | 2-8 |
| 3.0 VA CURRENT USAGE OF MUMPS | 3-1 |
| 3.1 Decentralized Hospital Computer Program | 3-1 |
| 3.2 VA Software Development | 3-10 |
| 3.3 VA Current Usage | 3-13 |
| 4.0 VA POSITION ON MUMPS | 4-1 |
| 4.1 VA Evaluation of MUMPS Language | 4-1 |
| 4.2 VA Proposed Enhancements to MUMPS | 4-5 |
| 4.3 VA Commitment to MUMPS and DHCP Development | 4-6 |
| 4.4 Summary | 4-7 |
| 5.0 SUMMARY | 5-1 |
| 5.1 MUMPS Programming Language | 5-1 |
| 5.2 VA Usage of MUMPS | 5-2 |
| 5.3 VA Evaluation of MUMPS | 5-2 |
| 5.4 VA Position on MUMPS | 5-3 |

TABLE OF CONTENTS

(Continued)

APPENDIXES

- A — MUMPS Survey**
- B — MUMPS Survey Responses**
- C — MUMPS Interview Questionnaire**
- D — MUMPS Interview**
- E — Abbreviations**
- F — References**

LIST OF EXHIBITS

| Number | | Page |
|--------|--|------|
| 2-1 | Distinctive Characteristics of MUMPS | 2-1 |
| 2-2 | MUMPS Symbolic Operators | 2-3 |
| 2-3 | One-Character Commands in MUMPS | 2-4 |
| 2-4 | The Tree Structure of MUMPS Arrays | 2-7 |
| 2-5 | Timeline of MUMPS Standardization | 2-9 |
| 2-6 | The MUMPS Standards Process | 2-11 |
| 3-1 | DHCP Architecture | 3-11 |

EXECUTIVE SUMMARY

The Office of Technical Integration has requested a study of the Massachusetts General Hospital Utility Multi-Programming System (MUMPS) programming language and Department of Defense (DoD) Automated Information Systems (AISs) written in MUMPS that support the medical business area.

As a component of this study, this report examines the Department of Veterans Affairs (VA), which has used MUMPS to develop and maintain several large AISs operating in VA hospitals and facilities. The VA has been active and influential in developing and using the MUMPS programming language and has become a leader in the MUMPS community. The VA's current use of MUMPS, evaluation of the language, and position on its future use within the agency are important considerations in assessing the language.

MUMPS is an interactive programming language and general-purpose data base management system (DBMS) that has been used widely in the development of medical information systems. MUMPS was developed by the Laboratory of Computer Science at the Massachusetts General Hospital in 1967. Although MUMPS is not used widely throughout all applications, it has had a prominent role in the development of health information systems (HISs) because of its origins. Versions of MUMPS are available for mainframes, minicomputers, workstations, and personal computers. The language is best known for its ability to handle unstructured textual information, its data sharing capabilities, and its interpretive nature.

MUMPS is an American National Standards Institute (ANSI), Federal Information Processing Standard (FIPS), and International Standards Organization/International Electrotechnical Commission (ISO/IEC) Standard language standard. The VA has actively sponsored the continued enhancement of the MUMPS language standard.

The VA has developed the Decentralized Hospital Computer Program (DHCP), an enterprise-wide health information system HIS that is used in 168 of the 172 VA hospitals and 12 clinics. Since its inception in 1982, the VA has invested from \$800 million to \$1 billion in DHCP. DHCP applications cover the spectrum of hospital services, from patient registration and admission to clinical applications and administrative functions. The DHCP software is modular and provides excellent portability between hardware platforms. MUMPS is used as the sole development language for DHCP. Ancillary tools are used for advanced statistics and reporting to extend DHCP functionality in areas in which MUMPS is currently deficient.

The VA finds MUMPS to be an excellent fit for the functional requirements of DHCP because of the language's following strengths:

- Exceptional string handling
- Powerful operators and functions
- Interpretive operation
- Rapid prototyping
- Advanced query capabilities/pattern matching
- Integrated data base system

However, the VA finds that MUMPS currently provides poor support for advanced arithmetic and statistical functions. The Oracle Relational Data Base Management System (RDBMS) and Statistical Analysis System (SAS) statistical software are used to enhance DHCP in these areas.

The VA has made a long-term commitment to the MUMPS environment, intending to use the language for DHCP development and maintenance for at least the next 10 years. Cost studies performed by the VA in the mid-1980s have shown MUMPS to be cost effective for continued development of DHCP programs. The VA considers MUMPS to be superior to other languages in productivity, maintainability, ease of learning, and portability.

1.0 INTRODUCTION

The Office of Technical Integration has requested a study of the Massachusetts General Hospital Utility Multi-Programming System (MUMPS) programming language and Department of Defense (DoD) Automated Information Systems (AISs) written in MUMPS that support the medical business area. This report focuses on one aspect of the study: the VA's current use, evaluation, and position on future use of the MUMPS programming language.

1.1 Study Background

The VA is one of the largest users of the MUMPS programming language and has a repository of over 40 AIS modules developed and maintained using MUMPS. This study will report on the VA's position with regard to the following:

- The use of MUMPS to maintain existing systems and develop new systems
- The advantages and disadvantages of using MUMPS in a development and maintenance environment
- An evaluation of the features of MUMPS and proposed extensions to the 1990 ANSI Language Specification

The following sections describe the methods used to gather information as well as the organization of the report.

1.2 Data Gathering

The information presented in this report was gathered by canvassing experts within the VA concerning both the current and future use of the MUMPS programming language. Additional information was gathered from the MUMPS Users Group (MUG), the MUMPS Development Committee (MDC), reference materials on MUMPS-based AISs, and the MDC's MUMPS Language Standard (1990 and 1993 ANSI).

1.2.1 MUMPS EXPERTS

To gather information for this study, VA experts were consulted to determine the VA's policy with respect to the MUMPS programming language. The initial contacts were made through telephone interviews. After these interviews, the experts were sent either follow-up surveys or preparatory question lists, depending on whether a face-to-face interview was anticipated. If a face-to-face interview with the expert was not planned, a short survey was sent to the expert

via facsimile. This survey allowed the expert to provide additional information not covered in the interview and to clarify information about how the VA uses MUMPS. A set of interview questions was forwarded prior to a face-to-face interview.

1.2.2 MUMPS SURVEYS

Surveys sent to VA experts who were not interviewed personally contained 13 questions designed to gather information on how MUMPS was implemented at the VA. The survey also sought to gather information comparing MUMPS to other programming environments and languages with which the respondent had experience. The MUMPS survey is included in appendix A. Appendix B contains the survey responses. Surveys were sent to Mr. Wally Fort, Mr. Lee Hirz, Ms. Maureen Hoye, and Dr. Cameron Schlehuber of the VA.

1.2.3 MUMPS INTERVIEWS

To confirm and clarify the VA's position on the MUMPS programming language, face-to-face interviews were conducted with two VA staff members (Mr. Javier Albarran and Dr. Ruth Dayhoff of the Washington, D.C., VA Information Systems Center (ISC)). As with the surveys, the interviews also sought to gather information about how MUMPS was implemented at the VA, and what the VA's current and future plans were with respect to MUMPS.

Before the interviews, questionnaires were sent to the VA staff members via facsimile to allow them to prepare their answers. The interview questionnaire is included in appendix C. Appendix D contains a transcript of the interview.

1.3 Document Organization

The remainder of this document is organized as follows:

- Section 2.0: Provides background information on the MUMPS programming language.
- Section 3.0: Describes the DHCP and the VA's current use of MUMPS.
- Section 4.0: Reports on the VA's evaluation of MUMPS and position on its future use in the agency.
- Section 5.0: Provides a summary of the report and preceding chapters.
- Appendix A: Contains the MUMPS survey used in collecting information.

- Appendix B: Contains the survey responses received.
- Appendix C: Contains the interview questionnaire.
- Appendix D: Contains a transcript of the VA interview.
- Appendix E: Lists the acronym definitions used in this report.
- Appendix F: Lists the references used in preparing this report.

2.0 MUMPS BACKGROUND

MUMPS was developed by the Laboratory of Computer Science at Massachusetts General Hospital in 1967. The objective was to develop an interactive, multi-user, minicomputer-based system that would support a hospital's special need to store, share, and manipulate textual data. Many of the characteristics of MUMPS were designed to support and manage the varied and limited data typical of the medical environment. Through the years, MUMPS has evolved into a general-purpose hierarchical data base management system that can support a wide variety of applications, such as scheduling, finance, and inventory.

2.1 Distinctive Characteristics of MUMPS

MUMPS has several characteristics that differentiate it from other programming languages and data base managers as shown in Exhibit 2-1, Distinctive Characteristics of MUMPS. The following sections discuss implications of these characteristics with respect to MUMPS' roles as a programming language and data base manager.

EXHIBIT 2-1: DISTINCTIVE CHARACTERISTICS OF MUMPS

- MUMPS is an interpreted language.
- Multiple, abbreviated statements can be included on a single line of program code.
- Shared data is available to other routines without program linkages or data declarations; temporary variable values are not removed from memory when routines terminate.
- MUMPS does not support declaration statements.
- MUMPS supports a persistent, shared data base; arrays are hierarchical and require no pre-allocation of memory; data on disk is stored in a sparse, hierarchical array.
- MUMPS variables are physically stored as variable-length character strings; they are converted when needed for numeric calculations.
- MUMPS allows the use of string subscripts as keys to array nodes; data is physically stored in sorted order according to subscript values.

2.1.1 MUMPS AS A PROGRAMMING LANGUAGE

Many of the programming characteristics of MUMPS are distinct from those found in other languages. MUMPS did not evolve out of the mainframe environment. Rather, the target platform of the language was a set of shared minicomputers operating in an interactive environment. The following sections describe the characteristics and implications of MUMPS as a programming language.

2.1.1.1 Interpretive Nature of MUMPS

MUMPS programs and source code are interpreted at run time. Source-code instructions are converted into machine-level instructions at the time of execution, rather than compiled before execution. The interpretive nature of MUMPS facilitates rapid prototyping, testing, debugging, and maintenance.

Rapid prototyping is facilitated because the program can be modified directly at run time. This saves the time required to compile a new version to determine whether it is acceptable.

Debugging is improved since the programmer can interact with the program at run time and execute commands directly at the keyboard. Information regarding the location of the error and the conditions that caused the error are immediately available.

Because MUMPS is an interpreted language, applications written in MUMPS tend to run more slowly than equivalent applications written in a compiled language. In order to execute each command, MUMPS must first translate the source-code instructions into machine instructions. Compiled programming languages perform this translation in advance and store the machine-level instructions as a program file that is executed at run time. Compiling programs improves efficiency since the translation of each statement occurs only once, not each time it is executed. In addition, compilers provide for faster execution.

In order to reduce the overhead associated with run-time interpretation, most implementations of MUMPS "tokenize" the code by partially compiling and optimizing the source code to improve execution speed. Tokenized MUMPS code still requires the MUMPS interpreter to operate, however. The dynamic structure of MUMPS at run time makes it nearly impossible to produce fully compiled MUMPS code because the compiler cannot predict the absolute memory addresses, required variable space, and instructions to be executed.

Despite these disadvantages, MUMPS programs can perform very effectively with textual data, especially where hierarchical storage is appropriate. A benchmark study by Alonzo reported that MUMPS surpassed certain FORTRAN and COBOL implementations for an application of this type.

2.1.1.2 Language Syntax

The syntax of MUMPS programs is typically terse and cryptic. MUMPS makes extensive use of symbolic operators, as shown in exhibit 2-2.

EXHIBIT 2-2: MUMPS SYMBOLIC OPERATORS

The line of code shown below uses the indicated operators to tell MUMPS to check the variable SPONSORID for a sequence of three numeric digits, and if it finds them, to write (after skipping a line) the variable value on column 20, with the variable inside the message "ID [sponsorid]: is valid.":

```
If SPONSORID?3N Write !,?20,"ID ",SPONSORID," : is valid."
```

The operators and their meanings are as follows:

| <u>operator</u> | <u>meaning</u> |
|-----------------|-------------------------------|
| ?3N | look for three numeric digits |
| ! | skip a line |
| ?20 | tab to column 20 |

MUMPS has a line orientation whereby several commands can be entered and executed from the same line of source code. Each line in a MUMPS program may have up to 255 characters. To support page and display width limitations, continuation of a line is indicated by three periods at the start of the next line. Most MUMPS commands can be abbreviated to the first letter of the command. Exhibit 2-3 is an example of MUMPS code using one-character commands, two or more to a line.

Combining multiple abbreviated statements on a line makes code easier to write but harder to read. Many MUMPS developers establish programming conventions and standards to enhance the maintainability of the code through adequate program documentation and proper use of MUMPS syntax.

2.1.1.3 Routine Structure

The MUMPS Portability Standard (described in section 2.2.1) limits the collective size of MUMPS routines and local variables executing in main memory to 4,000 bytes. The actual allowable routine size is usually implementation specific. The allowable size of a routine depends on the operating system and hardware issues such as available memory, buffers, disk caches, and other jobs.

EXHIBIT 2-3: ONE-CHARACTER COMMANDS IN MUMPS

The following MUMPS code fragment creates an array of numbers from 1 to 10, searches the array, and writes the highest and lowest values encountered in the array:

```
For A=1:1:10 Set ARY(A)=A
S HI=0, LO=999999
F A=1:1:10 S:ARY(A)<LO LO=ARY(A) S:ARY(A)>HI HI=ARY(A)
W !,"Highest value = ",HI,!,"Lowest value = ",LO
```

Note the use of one-character command abbreviations:

S for Set

F for For

W for Write

MUMPS reads routines from disks or disk buffers when routines are called to execute. Routines are swapped between main memory and disk storage as needed, and are removed from memory upon termination. The temporary variable space remains unchanged, allowing other routines access to temporary variables. Program linkage declarations are not supported, since routines in a program are not loaded before execution.

The MUMPS routine structure has several implications for the developer:

- Local variables are available for other routines, eliminating the need to explicitly declare the variables.
- MUMPS does not automatically protect the programmer from unintentionally reusing local variables.
- The programmer must explicitly reinitialize local variables to prevent their unintentional reuse.

2.1.1.4 Declarative Statements

MUMPS does not support declarative statements for constants, data structures, variables, procedures, or functions. The following characteristics contribute to the lack of declarative statements in MUMPS:

- MUMPS is a command-line-oriented language. Declarations are defined at the time the function is required.

- MUMPS does not run in compiled mode, so linkages between separately compiled modules are not necessary.
- MUMPS variables and arrays are automatically available to all routines operating in the current process.
- MUMPS variables are physically stored as variable-length character strings.

The elimination of declarative statements in MUMPS is a major departure from other languages. Although the elimination of declarations may make programs easier to write, the use of declarations helps to clearly define the roles of variables and procedures in a program. Use of declarations can also promote early detection of typographical errors. The absence of declarations can reduce maintainability, since declarative statements help to document a program.

2.1.2 MUMPS AS A DATA BASE MANAGEMENT SYSTEM (DBMS)

MUMPS was designed as both a general-purpose programming language and a DBMS for medical information systems. The principal functions of a medical information system are to store, retrieve, and manipulate shared textual data. Because of its integrated data base system, MUMPS stores and manipulates data differently from other programming languages.

2.1.2.1 File Structures

The MUMPS file structure is a sparse hierarchical array implemented in a tree structure. MUMPS arrays are persistent; that is, they exist even after the routine that created them has stopped running. Local variables are shared by all applications using the same directory, so MUMPS provides constructs that the programmer can use to prevent the reuse of local variables.

Most programming languages implement a sequential file structure in the form of a matrix. The matrix structure provides advantages in performing advanced statistical and mathematical functions. However, these file structures are usually static and hard to adapt to changing requirements. Selecting individual records from a MUMPS hierarchical file is very efficient compared to sequential matrix files.

2.1.2.2 Data Types

Most DBMSs support certain basic data types, such as integer and character, and abstract data types, such as date, time, and currency. MUMPS does not support any data types explicitly. All data is stored physically as variable-length character strings. MUMPS automatically evaluates each variable according to the operation being performed. A numeric operation will return a numeric data type; a character operation, such as a concatenation, will return a character-string

type. MUMPS will evaluate a variable as either a character string, integer, floating-point number, or Boolean value, depending on how it is to be used.

Because numbers are handled as text strings, they must be converted into an internal numerical form when used in calculations. These conversions are not necessary for languages that support integer and floating-point numbers. Thus MUMPS is much less efficient at complex numerical calculations than a language that supports machine numbers. In addition, MUMPS requires additional space to store numbers since it does not use packed machine-level representations.

MUMPS is essentially a nontyped language, so a programmer has an unlimited set of operations that may be performed on data. However, data typing is useful in documenting the nature of an element, domains of its acceptable values, and logical operations that can be performed on it.

2.1.2.3 Arrays

Arrays (referred to in MUMPS as "globals") have no predetermined fixed or allocated size. Records in an array are variable length. Variable names may have up to 31 characters, allowing the programmer to describe the data completely; however, MUMPS uses only the first 8 characters to distinguish between variables.

Most languages implement a matrix array structure with a predefined number of columns and rows. The variable-length record structure of a MUMPS sparse hierarchical array makes efficient use of storage space. Disk storage requirements are determined by the amount of data physically stored, not by a predefined array size and record length; null values and redundant trailing spaces are not saved.

The branches in the hierarchical array structure are termed "subscripts." Subscripts are the keys to the data contained in each node in the array. Each subscript identifies a branch in the array, corresponding to a level in the array hierarchy. The physical ordering of data in an array is determined by the subscript values. Instead of storing data in the order it is entered, MUMPS stores data in sorted order according to the subscript values. Subscript values may be integers or character strings of up to 31 characters.

Storing in order of subscript values improves the search and retrieval ability of MUMPS. For example, a scan of records with the last name starting with the letter "G" will begin sequentially with the first record beginning with "G," and terminate when it encounters a name beginning with the letter "H."

MUMPS arrays and subscripts are defined at the time a function uses the array. Exhibit 2-4, The Tree Structure of MUMPS Arrays, illustrates the hierarchical array structure of MUMPS.

The MUMPS hierarchical array structure preceded the relational table model. As in the relational model used in RDBMSs, the data is normalized and redundant data is not stored within the

EXHIBIT 2-4. THE TREE STRUCTURE OF MUMPS ARRAYS

| <u>ARRAY</u> | <u>FIRST SUBSCRIPT (SSN)</u> | <u>FIRST SUBSCRIPT (DATE)</u> | <u>VALUE</u> |
|--------------|--------------------------------------|---------------------------------------|--------------|
| PATREC | 372-40-1234 | | |
| | | | |
| | 372-40-1349 | 12/1/91 | "John Smith" |
| | | 1/1/92 | "Ultrasound" |
| | | 1/23/92 | |
| | 372-40-2299 | | |
| | | | |

The following statements assign the values shown in the boxes indicated in **bold** above:

```
Set ^PATREC(372-40-1349)="John Smith"
Set ^PATREC(372-40-1349,010192)="Ultrasound"
```

system. Tables and attributes in RDBMSs are explicitly defined in advance of their use; in MUMPS, the tables and attributes are defined implicitly when they are used. The MUMPS sparse hierarchical structure is most useful for storing textual or other data with records of varying length. User views of the data in an RDBMS must be defined explicitly through the relationships between table values; however, hierarchical data bases implicitly contain relationship information indicating that one record is dependent upon another.

Unlike the relational model, entity relationships are not established by matching values in key fields; instead, they are expressed through the hierarchy of the tree. This convention has the following implications:

- A relational data base is more flexible than a hierarchical data base, since tables can be related to each other in arbitrary ways, as opposed to only through dependency.
- A hierarchical data base is much faster to use for certain applications, since the records are implicitly related, and two sets of data do not have to be inspected and compared to locate matching values for each join.

Different user views of a MUMPS array would require the view to be defined at the physical level, storing an additional array with redundant subscripts and nodes. The inability to support different user views from the same physical structure is a significant drawback. If many user views are necessary, the storage efficiency of variable-length record structures is largely negated by the additional redundant data.

2.2 MUMPS Standardization

By the early 1970s, several incompatible dialects of MUMPS had evolved. The widespread incompatibility severely hampered the portability of MUMPS applications between environments. The MUMPS Development Committee (MDC) was subsequently established to define a MUMPS Language Standard to be submitted to the American National Standards Institute (ANSI). The MDC is responsible for revisions to the MUMPS Language Standard and evaluates proposals to modify or enhance the language.

2.2.1 THE MUMPS DEVELOPMENT COMMITTEE

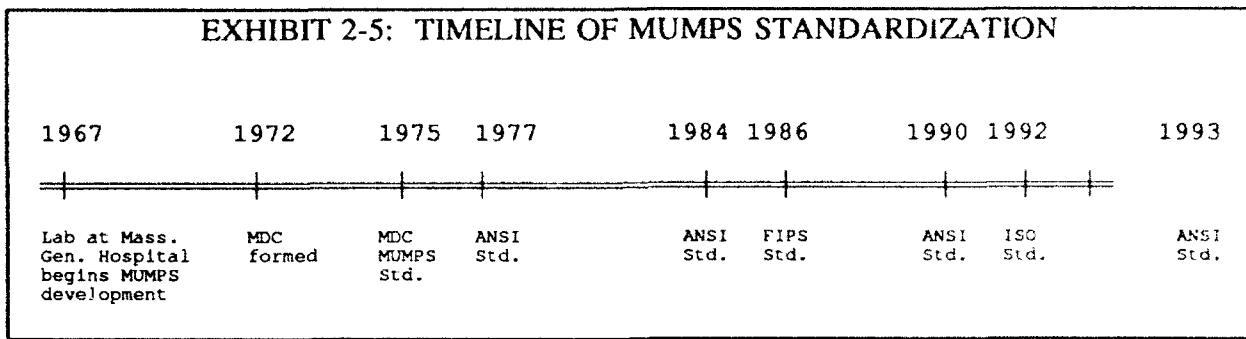
The MDC is composed of subcommittees that address general functional areas of the language, such as user interface or data management. Each subcommittee is further organized into task groups that are responsible for a specific aspect of the language, such as X-Windows or string handling. Any organization may request MDC membership; so far, no requests have been denied.

Member organization representatives may also join task groups in which they have an interest. The MDC currently includes approximately 10 vendor and 50 user organizations. The member organizations send representatives to the three conferences that the MDC holds each year. Much of the conference time is spent working in task groups. The entire MDC meets at the conclusion of the conference. The VA has provided a grant to pay for the annual meeting of the MDC; the DoD and Indian Health Service (IHS) have also periodically provided funds. The MDC is also funded through annual membership dues of \$100.

The MDC has continued to be responsible for maintaining and developing the MUMPS Language Standard. MUMPS has been approved as an ANSI Standard (ANSI/MDC X11.1-1990), a Federal Information Processing Standard Publication (FIPS PUB 125), and an International Standards Organization/International Electrotechnical Commission Standard (ISO/IEC 11756). Exhibit 2-5, Timeline of MUMPS Standardization, outlines the various stages of MUMPS development and standardization.

The MDC has published both a MUMPS Language Standard and a MUMPS Portability Standard. The ANSI Language Standard guides implementors in the minimal requirements that must be met in order to conform to the ANSI Standard. The Portability Standard is a subset of the ANSI Language Standard, serving implementors who create MUMPS interpreters as well as developers

EXHIBIT 2-5: TIMELINE OF MUMPS STANDARDIZATION



who create MUMPS applications. Each standard is interpreted as a minimum specification to be met by implementors and a maximum specification to be exercised by developers. If both the implementor and developer adhere to the Portability Standard, the portability of the MUMPS program to other platforms will be assured. The existence of the Portability Standard has been a great benefit to MUMPS users. Both MUMPS developers and end users of MUMPS applications are able to take advantage of the higher performance and lower costs associated with newer technology.

The MUMPS Portability Standard means that program developers who adhere to the standard can be confident that their programs will run successfully on any MUMPS implementation. This contrasts with programming languages that lack portability standards. In FORTRAN, for example, a program may comply with the language standard but fail to execute because the use of arrays exceeds the memory capacity of the target machine.

Implementations of MUMPS may contain extensions to the ANSI Standard. Some implementation-specific extensions support operating system and hardware-dependent utilities. Other extensions to the current ANSI Standard include official Type A Extensions that the MDC has approved for inclusion in future standards in the process described in the following sections.

2.2.2 MUMPS STANDARDIZATION PROCESS

Anyone can submit proposals for extensions or modifications to the MUMPS Language Standard. The MDC is responsible for reviewing and approving all such proposed extensions or modifications. The MDC makes the ultimate determination as to whether or not the proposal is adopted as an extension or included in the next submission of the MUMPS Language Standard to ANSI.

2.2.2.1 Standardization by MDC

Exhibit 2-6, The MUMPS Standards Process, shows the MDC's formal process for adopting modifications to the MUMPS Language Standard. The MDC is organized into subcommittees. Each subcommittee is responsible for several task groups. The MDC assigns requests for

modifications to the appropriate subcommittee, which in turn is responsible for assigning the request to an appropriate task group or forming a new task group if one does not already exist.

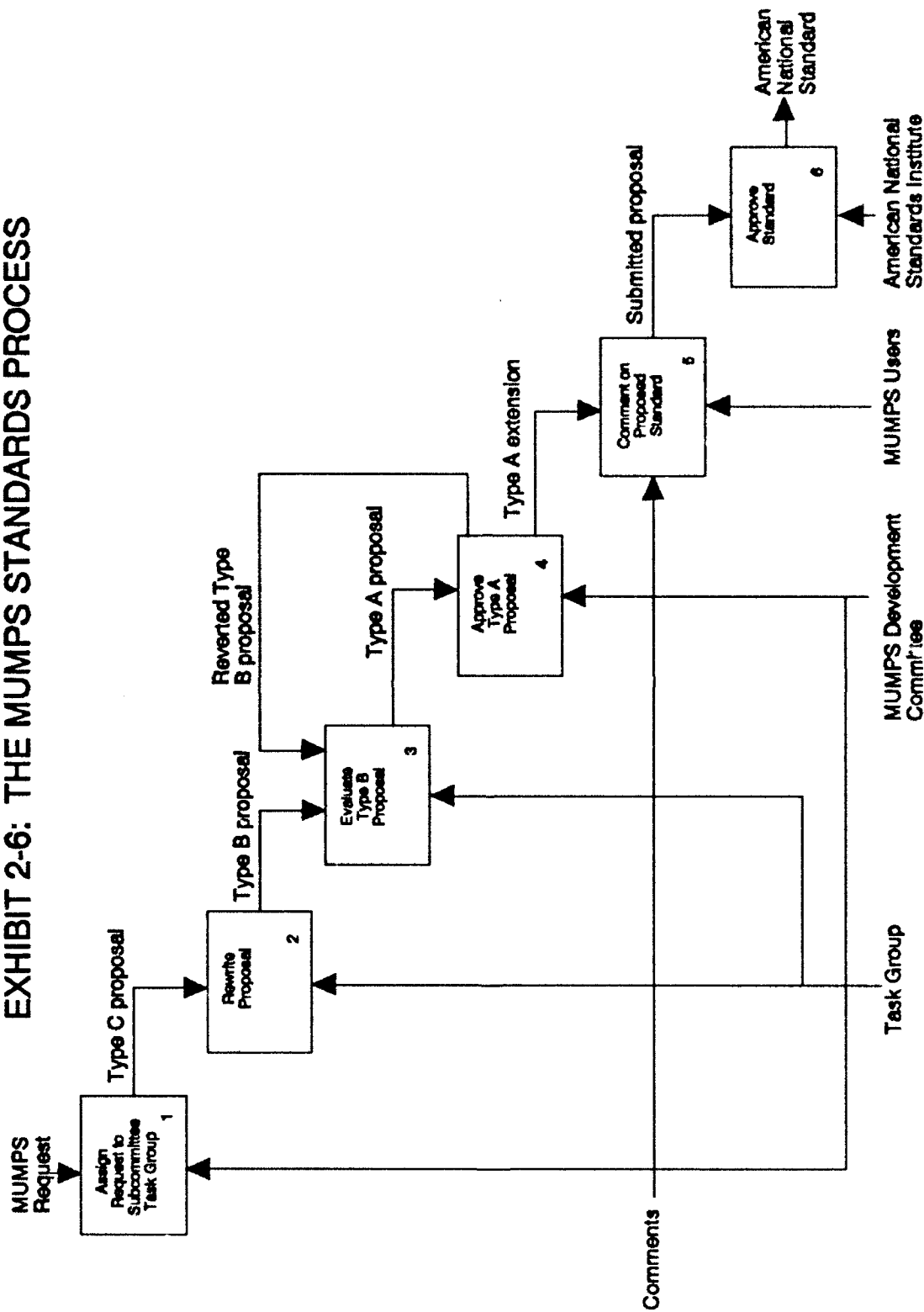
A request at this stage is termed a "Type C proposal." The proposal will be rewritten in a formal manner and, if deemed worthy of further consideration, it is termed a "Type B proposal." When the proposal is deemed complete enough to be submitted to the full MDC for consideration, it becomes a "Type A proposal." When a proposal receives Type A status, no other changes to the proposal document will be accepted. If it is determined that changes are necessary, the proposal reverts to Type B status and is returned to the subcommittee for modification.

2.2.2.2 Ratification by ANSI

Type A proposals that MDC approves receive "Type A extension" status. All Type A extensions are included in the next submission of the MUMPS Language Standard to ANSI. The MDC seeks comments from the MUMPS community on MDC Type A extensions to the Language Standard, and these comments are submitted to ANSI along with the MDC MUMPS Language Standard. ANSI will approve the MDC MUMPS Language Standard when it is satisfied with the MDC responses to questions on specific proposals that arise from the canvassing process. The main role of ANSI in the standardization process is to ensure that the MDC has considered the material interests of all affected parties in preparing the MUMPS Language Standard.

The active involvement of the MDC in the continual improvement of the MUMPS Language Standard has been significant in the general portability of MUMPS applications. Proposed extensions to the Language Standard are publicized widely. The MDC actively seeks feedback from MUMPS users on proposed extensions to the ANSI Language Standard. This collaboration between the standards-setting body and the user community has led to a MUMPS Language Standard to which both MUMPS implementors and application developers closely adhere. Vendor and user consensus over weaknesses in the ANSI Language Standard contributes to future revisions to the standard.

EXHIBIT 2-6: THE MUMPS STANDARDS PROCESS



Each subsequent revision to the ANSI Language Standard has addressed specific vendor and user concerns. Changes to the Language Standard that are not compatible with previous versions are adopted cautiously. The most significant events and changes in the MUMPS Language Standard include the following:

- 1977 Standard — Merged the several incompatible dialects of MUMPS that had evolved into a single standard
- 1984 Standard — Implemented the use of descriptive subscripts in MUMPS arrays
- 1990 Standard — Provided support for structured programming techniques
- 1993 Standard (proposed) — Will provide support of MUMPS bindings to industry standards (Structured Query Language (SQL), X-Windows, and Graphical Kernel Services (GKS))

2.2.2.3 FIPS Adoption by NIST

The National Institute of Standards and Technology (NIST) follows a standard process in recommending the establishment or revision of a standard to the Secretary of Commerce. A sponsoring organization, in this case the MDC, submits a standard to NIST for consideration. NIST then elicits comments on the standard from interested or affected parties. Comments are accepted for approximately 3 months. NIST then reviews comments and prepares responses. The sponsoring organization may provide technical background to support NIST's response to the comments. When all comments have been addressed to NIST's satisfaction, the standards are forwarded to the Secretary of Commerce for approval.

After the approval of the Secretary of Commerce, NIST issues a FIPS PUB specifying the standard. The FIPS PUB becomes effective 6 months after the date of approval. A 12-month implementation period commences on the effective date of the standard. This implementation period gives implementors and developers time to adjust to the adoption of the new standard. At the end of the implementation period, Government usage of the standard becomes mandatory.

As usual, FIPS PUB 125 follows the form of the current ANS for MUMPS. It is not uncommon for a FIPS PUB to be superseded by a recently revised ANS. In this case, the FIPS PUB and the ANS differ until the FIPS PUB is revised.

3.0 VA CURRENT USAGE OF MUMPS

The VA is a major user of the MUMPS programming language for its medical information systems. The VA has been developing its medical information system, the Decentralized Hospital Computer Program (DHCP), in MUMPS since 1982, and has installed it at 168 of the 172 VA hospitals nation wide. The VA has ported its MUMPS application software to a number of hardware platforms, allowing them to use the hardware with the best cost-to-performance ratio. MUMPS has been used by the VA for all DHCP development and maintenance. Modules of DHCP have been made commercially available to other health organizations attempting to implement medical information systems.

3.1 Decentralized Hospital Computer Program

In the late 1970s and early 1980s, the VA decided to automate its facilities. A few local pilot projects had been developed to support a single activity, such as laboratory results or patient registration, admission, discharge, and transfer. However, none of the projects held the potential to become a national system.

In 1982, the VA decided to integrate and standardize its information systems by developing and implementing a national medical information system, DHCP. After a demonstration of a MUMPS pilot project, the VA selected MUMPS as the programming language for DHCP. At the time, relational DBMS technology was not considered mature enough for enterprise computing, whereas MUMPS provided acceptable performance as both a DBMS and a general programming language.

By 1985, the core modules of DHCP had been developed and implemented. This core consisted of the following modules:

- Patient Registration, Admission, Discharge, and Transfer
- Clinic Scheduling
- Inpatient/Outpatient Pharmacy
- Clinical Laboratory

DHCP is composed of a "Kernel," system modules, clinical application modules, and management application modules, and has been deployed to 168 VA Medical Centers (VAMCs).

3.1.1 KERNEL

The Kernel provides a bridge between DHCP application modules and the underlying hardware or operating system. In effect, the Kernel replaces the operating system by handling all system functions (for example, screen display and keyboard input) for the application module. Using

the Kernel as a bridge greatly increases portability among different hardware platforms, operating systems, and MUMPS implementations, since the applications need to interact only with the Kernel system. In addition, the user interface, development environment and tools, and management system are standardized. This greatly reduces software life cycle costs since costs for software development, training, and maintenance are reduced. Furthermore, the costs of application conversion are eliminated when changing hardware or operating systems because the Kernel remains the same; otherwise, many of the module's input and output routines would have to be modified or rewritten.

The Kernel comprises of the following components:

- Security Management
- Device Handler
- FileMan
- MailMan
- MenuMan
- TaskMan
- Tools

Kernel services support individual DHCP modules through security and auditing, device management, data base management, electronic mail, menu management, background task management, and programming utilities, as well as system management utilities.

3.1.1.1 Kernel Security Management

The Kernel security management features control user access to system resources. Access to programs, menu options, files, data records, data fields, and devices can be controlled on the basis of the user, date, and time. The Kernel can provide audits of any system resources.

3.1.1.2 Kernel Device Handler

The Kernel device handler defines the interfaces for various devices, such as terminals and printers. The device handler also insulates the rest of DHCP from operating system-specific and hardware device-specific codes by providing uniform interfaces.

3.1.1.3 Kernel FileMan

The Kernel FileMan is the DBMS for DHCP. The FileMan provides a DBMS that supports file-level and field-level data security, ad hoc data base queries, data base modification, and an active data dictionary. FileMan also provides a software development environment with tools for

screen-oriented data entry and display, templates for data entry and reports, templates for data queries and sort criteria, and screen editor support.

3.1.1.4 Kernel MailMan

The Kernel MailMan is the general data transfer system for DHCP. MailMan provides comprehensive electronic mail, teleconferencing, and data transmission services, as well as electronic notification of individuals and groups.

3.1.1.5 Kernel MenuMan

The Kernel MenuMan manages all menus used in DHCP. Menus can be customized for individuals and can share or restrict menu options.

3.1.1.6 Kernel TaskMan

The Kernel TaskMan handles all background tasks for DHCP, such as print queues. Tasks can be controlled by their users specifying the device, priority, and time of execution.

3.1.1.7 Kernel Tools

The Kernel tools assist programmers and system managers in configuring applications for local requirements, monitoring system status, monitoring usage and performance, and purging and maintaining files.

3.1.2 SYSTEM MODULES

The VA has developed system modules that support all Kernel applications. The following system application modules have been developed to support DHCP:

- LetterMan
- ClassMan
- DoD Contingency Plan Reporting
- Generic Code Sheet

The system modules provide a standard document editor, on-line instruction, reporting of bed availability, and code sheet entry.

3.1.2.1 LetterMan

LetterMan is the standard FileMan word processor/editor. LetterMan allows the user to create, format, and edit documents. LetterMan provides standard editing functions such as tabs, text highlighting, headers and footers, block cut and paste, spell checking, and macros.

3.1.2.2 ClassMan

ClassMan provides on-line classroom instruction support and slide show capabilities through utilities and interactive audiovisual communications. ClassMan can be run interactively by an instructor or with a prerecorded training tape.

3.1.2.3 DoD Contingency Plan Reporting

DoD Contingency Plan Reporting allows the DoD to use VA beds. A national data base of available beds is arranged by category, and can be accessed manually or through a link to the Defense Medical Regulating Information System (DMRIS).

3.1.2.4 Generic Code Sheet

The Generic Code Sheet module is designed to eliminate keying errors by allowing data to be entered and transmitted electronically from the medical service to the national data base. The code sheet module contains about 250 generic code sheets, and has tools to develop new code sheets. The code sheet module provides security to prevent unauthorized access.

3.1.3 CLINICAL MODULES

Each of the clinical application modules supports a specific clinical area, although the modules are integrated through the Kernel's utilities and the shared MUMPS data base. The following clinical application modules have been developed for DHCP:

- Clinical Record
- Dentistry
- Dietetics
- Immunology Case Registry (ICR)
- Laboratory
- Medicine
- Mental Health
- Nursing
- Oncology

- Pharmacy
- Physician Data Query (PDQ)
- Quality Management
- Radiology
- Social Work
- Surgery

3.1.3.1 Clinical Record

The Clinical Record package is a pilot project at several ISCs to test the full use of automation for clinical activities. The Clinical Record contains several modules that integrate data from the various DHCP modules for clinical analysis. Current applications include Order Entry/Results Reporting, Health Summary (Customized), Allergies/Adverse Reactions, Problem Index, Progress Notes, Generic Text Generator, Consults, and Clinical Observations/Measurements.

3.1.3.2 Dentistry

The Dentistry module supports the maintenance of patient dental records and treatment data, transmits reports to the VA central data base, displays reports in various formats, and integrates with the Medical Administration Service (MAS) scheduling module to schedule appointments with critical path assistance.

3.1.3.3 Dietetics

The Dietetics module supports the ordering of dietetic services; performance of nutritional analysis, screening, and assessments; management of recipes and ingredients; and various reporting functions.

3.1.3.4 Immunology Case Registry (ICR)

The ICR module maintains local and national registries (lists) for monitoring disease categories. Human Immunodeficiency Virus (HIV) registry reports are generated for submission to the Centers for Disease Control (CDC). The ICR integrates with DHCP in obtaining relevant patient treatment information.

3.1.3.5 Laboratory

The Laboratory module assists the laboratory/pathology service in the management of department workload and reporting processes. Several modules have been implemented to support several

functional areas, such as Anatomic Pathology, Blood Bank, Phlebotomy/Ordering, and Microbiology.

3.1.3.6 Medicine

The Medicine package consists of several modules to support the entry, editing, and viewing of data from various medical tests and procedures. Current modules implemented include Cardiology, Pulmonary, Gastrointestinal, Hematology, Pacemaker, and Oncology. Future modules to be implemented are Renal/Dialysis, Rheumatology, Infectious Disease, Endocrinology, and Dermatology.

3.1.3.7 Mental Health

The Mental Health module supports clinicians in the efficient diagnosis of health and medical problems. Psychological tests and interviews, clinical records (such as patient and treatment data and inquiries), and clinic and ward management functions are supported.

3.1.3.8 Nursing

The Nursing module contains Clinical, Administration, Education, and Quality Assurance modules that support patient care, continuous service monitoring (for example, vital signs), and nursing management reports and queries. Future enhancements to the package will provide the integration of the Clinical module with DHCP Order Entry/Results Reporting, Staff Scheduling, Patient Assessments, Progress Notes, Discharge Planning, Patient Education, and Clinical Monitors.

3.1.3.9 Oncology

The Oncology module supports tumor registry personnel in abstracting cancer cases, scheduling follow-up work on cancer patients, maintaining and generating cancer registry listings, and producing the Hospital Annual Report.

3.1.3.10 Pharmacy

The Pharmacy package consists of several modules that manage the inventory/stock of drugs, monitor inpatient medications, standardize drug information, and manage the outpatient pharmacy.

3.1.3.11 Physician Data Query (PDQ)

The PDQ is a computerized data base of cancer treatment information developed by the National Cancer Institute, Department of Health and Human Services, with the assistance of cancer experts nation wide. PDQ contains information on treatments, ongoing clinical trials and research studies, doctors who treat cancer, and hospitals with cancer programs.

3.1.3.12 Quality Management

The Quality Management package includes several modules that track the verification of clinical staff credentials and privileges; monitor reports and recommendations from the Office of Inspector General (IG), the General Accounting Office (GAO), the Joint Commission on Accreditation of Healthcare Organizations (JCAHO), and other external review organizations; and compile data on patient incidents for reporting and tracking by medical centers and the national Quality Assurance data base. These modules also identify events that require follow-up review, generating worksheets for reviewers and identifying problems; allow the VA to gather comparative data for continuous improvement in health care; and automate the tracking of patient care, admission, length of stay, and diagnoses.

3.1.3.13 Radiology

The Radiology module automates order entry of radiology requests, registration of patients for examinations, processing of examinations, and recording of radiology reports and results. The Radiology module is integrated with other DHCP modules to provide access to patient medical and laboratory test histories; it also monitors film usage.

3.1.3.14 Social Work

The Social Work module assists social workers in case management, identification of patients likely to require social work assistance, and maintenance of the network of community social work agencies for patient referrals.

3.1.3.15 Surgery

The Surgery module supports the scheduling and use of operating room resources and personnel; monitors surgical cases by tracking information such as diagnosis, type of surgery, and outcome; and generates various management reports for the chief of surgery and other medical and administrative staff.

3.1.4 MANAGEMENT MODULES

The VA has developed management modules that help track finances, information, and scheduling. The VA has developed the following management application modules to support management staff:

- Decision Support System (DSS)
- Diagnostic Related Group (DRG) Grouper
- Engineering
- Integrated Funds Control, Accounting, and Procurement (IFCAP)
- Interim Management Support (IMS)
- Library
- Medical Administration
- Medical Care Cost Recovery (MCCR)
- Operation Desert Storm (ODS)
- Patient Data Exchange (PDX)
- Personnel and Accounting Integrated Data (PAID)
- Record Tracking

3.1.4.1 Decision Support System (DSS)

The DSS module extracts and captures data from other DHCP modules and transmits the data to any number of internal or external decision support systems. The decision support systems provide patient-level costing, case-mix management, department and hospital budgeting, and variance analysis.

3.1.4.2 Diagnostic Related Group (DRG) Grouper

The DRG Grouper classifies patients of similar age, gender, diagnoses, operation codes, and discharge status into a single DRG. The DRG Grouper is based on the Medicare Grouper, and provides annual updates that conform to the Health Care Finance Administration (HCFA) Medicare Grouper.

3.1.4.3 Engineering

The Engineering module manages work order information for engineering sections, equipment repair histories, preventive maintenance schedules, construction project management, space management, and accident reporting.

3.1.4.4 Integrated Funds Control, Accounting, and Procurement (IFCAP)

The IFCAP module automates the distribution of funds, tracks accounts receivable and inventories, allows patients to manage their personal funds, and processes purchase orders and receipts.

3.1.4.5 Interim Management Support (IMS)

The IMS module generates reports for a VAMC's management. The module tracks staffing, finances, machines and equipment, space resources and construction, station and service workload, and quality of care.

3.1.4.6 Library

The Library module automates control of serial publications for VAMC library staff. The module maintains a serials catalog, maintains vendor and purchasing information, categorizes serials by type and subject, lists serials, and tracks circulation of serials.

3.1.4.7 Medical Administration

The Medical Administration package is composed of a set of modules that support patient registration, admission, discharge, and transfer; exchange veteran information between Veteran Benefits Administration regional offices and VAMCs; track incomplete and delinquent patient records; automate fee basis compensation; determine veteran eligibility status; and schedule patient appointments.

3.1.4.8 Medical Care Cost Recovery (MCCR)

The MCCR package is a set of modules that support pharmacy co-payment collection and third-party billing.

3.1.4.9 Operation Desert Storm (ODS)

The ODS module was developed to support the treatment of casualties resulting from the Persian Gulf war. VAMCs used ODS for the electronic registering of casualties in a national data base, remote data entry, and reporting to the VA Central Office.

3.1.4.10 Patient Data Exchange (PDX)

The PDX module requests and receives patient demographic, episodic care, and diagnostic information between medical centers to create a composite patient record. PDX also processes and transmits requests for patient information.

3.1.4.11 Personnel and Accounting Integrated Data (PAID)

The PAID module automates the creation and transmission of personnel and fiscal transactions to VA data processing centers. It also automates the transmission of VA payroll data to VA payment centers.

3.1.4.12 Record Tracking

The Record Tracking module tracks medical records and radiology films. Record creation, deactivation, reactivation, transfer, and deletion are automated.

3.2 VA Software Development

In developing DHCP, the VA established the following criteria for software design, development, and integration:

- Standardized and portable software
- Standard data elements
- Technical integration through a common data base, programming standards and conventions, and data administration functions
- Functional integration through common utilities
- Hardware and operating system independence
- Ease of use for both health care professionals and information resources manager
- System integrity and protection of data against loss and unauthorized change, access, or disclosure

To meet the software design, development, and integration criteria, the VA established policies and procedures that governed the entire software development process. These policies and procedures covered the entire life cycle for application software, ensuring that each application would work harmoniously with any combination of other applications.

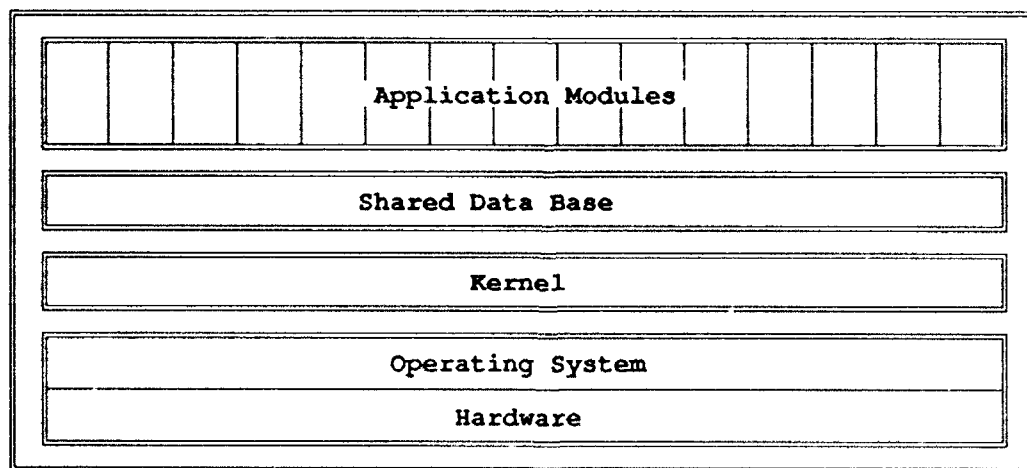
The VA chose a layered modular architecture for DHCP. All new clinical, management, and system modules had to be registered with the data base administrator. The code was required to conform to the 1990 ANSI Language Standard current at that time. Testing and validation procedures were established to ensure that the code met both functional and technical requirements for the VA. Release and distribution procedures were established to allow new code to be shared among the various VAMCs. Documentation standards that each module would have to meet were established. The strength of these standards greatly improved the portability of the VA's application code.

3.2.1 ARCHITECTURE

DHCP was designed with a layered modular architecture to insulate the application code from the underlying hardware and operating system. The user interacts with an individual application module that interfaces with the Kernel. The Kernel interfaces with the actual hardware and operating system. Exhibit 3-1 depicts the layered architecture of DHCP.

The layered architecture means that the Kernel essentially replaces the operating system by handling file management, message handling, task management, and other system functions. This architecture allows the VA to run all of its application software on any system to which it can port the Kernel.

EXHIBIT 3-1: DHCP ARCHITECTURE



The modular architecture specified by the VA also allows each VAMC to tailor its installation of application code modules. For example, a small clinic without a mental health ward would not install the Mental Health module, since the module would not be needed.

3.2.2 Module Registration

In 1982, the data base administrator at the Salt Lake City VA ISC was tasked with coordinating all VA software support. The data base administrator controls and assigns a "name space" to each module to prevent name conflicts with other modules. In addition, this registration prevents duplication of effort because development is coordinated.

A unique name space was assigned to each module of code by specifying a two- or three-letter tag that would be prefixed to each identifier within the module, such as a variable name or a routine name. For example, the "LAB" name space might have been assigned to the laboratory support module; every identifier within the module would have the "LAB" tag prefixed to it.

3.2.3 Data Management

The VA has developed FileMan to handle all data base management for its software. FileMan has an active data dictionary that is integrated with the MUMPS environment. FileMan is used for modifying, documenting, and querying the data base. FileMan also works with the data security module to provide data security and auditing.

3.2.4 Coding Standards

The VA established a number of coding standards that all software code is required to meet. The VA specified that the application code modules would interface only with the Kernel, not with the underlying operating system or hardware. In addition, the Kernel was the only part of DHCP that was allowed to use implementation-specific extensions or hardware-specific code.

All new VA code conforms to the current MUMPS Language Standard, which was approved by ANSI in 1990. The 1990 ANS added a number of improvements that greatly improved maintainability over the previous 1984 ANS. These changes improved the readability by making the language more structured and reducing the need for nonfunctional jumps between code segments. Section 3.0 provides a detailed description of these changes. In addition, the VA has developed code generators and other tools to automate the code development process.

3.2.5 Testing and Verification

The VA has established a testing and validation procedure for all software modules. After an application module is completed, it is sent to one Alpha site for testing and undergoes internal code verification by the developers. The Alpha site tests the module's functionality. The internal validation check verifies that the module's code meets the VA's technical standards.

After the Alpha site validates the module, and the developers have verified the code, the module is sent to three or four sites for Beta testing. It also undergoes external code verification by a site that did not develop the code. After the Beta testing and external verification are complete,

the code is ready to be distributed and is sent to a central office. The VA has developed software tools that check the level of conformance to VA standards for application code.

3.2.6 Documentation

When a module is released for general use within the VA, a set of standard documentation is written to accompany it. This set of documents includes a Clinical Manual, a User Manual, an Installation Manual, and Release Notes. The Clinical Manual and User Manual describe the functionality of the module. The Installation Manual contains information on installing the module, such as whether other modules must be installed before installing the module. The Release Notes describe any changes from previous versions of the module.

3.2.7 Software Distribution

The VA has developed software release procedures to control the distribution of software. A module must first pass the testing and validation procedures to ensure that it is compatible with other VA code, meets VA coding standards, and follows VA coding conventions. The module must be documented properly with a set of standard documentation. At that time a release letter is written, and the module becomes part of the central code library. Any VAMC can then electronically download the module from the code library and run the module. Although the entire DHCP system is available in the code library, the VA uses magnetic tape to distribute large modules or packages of modules.

3.3 VA Current Usage

The VA uses MUMPS as the development language for its medical information systems. The VA's primary medical information system, DHCP, is coded in MUMPS, with ongoing development spread out at seven locations nation wide. The VA programming staff continues to internally develop, maintain, and support DHCP software. The VA currently has a huge investment in DHCP software and hardware.

The VA also participates in joint development projects with the IHS and DoD. The VA also has tested new technologies by implementing commercial medical information systems at four VA hospitals, using them as a test suite for innovations that could be incorporated into DHCP.

3.3.1 VA INVESTMENT

The VA has made a significant investment in MUMPS technology over the past 10 years. Its current investment totals \$800 million to \$1 billion, \$400 million of which is for computer hardware. More specific information pertaining to the VA's investment in DHCP can be obtained from the VA's Medical Information Resources Management Office (MIRMO).

VA hospitals use MUMPS for applications spanning electronic mail and automated medical information systems. The VA has 168 hospitals and 12 regional facilities running DHCP. The VA has over 400 staff programmers in 7 VA ISCs. Each ISC is responsible for developing and maintaining a specific part of the overall code; for example, the Washington, D.C., VA ISC is responsible for part of the Kernel. Of the approximately 400 VA staff programmers, about 250 programmers develop application code or system Kernel code. Over 120 programmers maintain existing code, while the remainder assist users. With this large staff of programmers, the VA has strong MUMPS expertise, with the ability to internally support training, development, maintenance, and support functions.

3.3.2 JOINT DEVELOPMENT PROJECTS

The VA participates in joint development projects with DoD and the Indian Health Service (IHS). The DoD Health Affairs Office is in the process of implementing a large HIS, based mainly on DHCP, that will be deployed eventually at military hospitals world wide. The IHS uses several DHCP modules for their HISs.

3.3.2.1 DoD Health Affairs

In 1988, the DoD Health Affairs Office contracted with Science Applications International Corporation (SAIC) to develop its HIS, the Comprehensive Health Care System (CHCS). CHCS is based on DHCP, modified to meet DoD specifications. CHCS has been deployed at 2 Alpha site hospitals and 12 Beta site hospitals that completed Operational Testing and Evaluation (OT&E) in May 1992. CHCS was approved by DoD's Major Automated Information Systems Review Council (MAISRC) at the Milestone IIIA review on 19 May 1992. CHCS awaits congressional approval before worldwide deployment to all military hospitals. The current deployment plan is for the deployment of the basic functions of CHCS to all medical treatment facilities (MTFs) with 50 or more beds by 1995.

3.3.2.2 Indian Health Service

The IHS uses DHCP, with some modifications, to support its reliance on small outpatient clinics as opposed to inpatient hospitals. The IHS has been developing expert system queries. In addition, the IHS has been working on integrating data for the "whole patient." The IHS has developed MUMPS tools that scan modules to locate data areas that can be shared.

3.3.3 COMMERCIAL ALTERNATIVES

In addition to DHCP, the VA has been evaluating four full-scale commercial HISs at the Chicago Lakeside; Philadelphia; Brooklyn; and Big Spring, Texas, VA hospitals under the Integrated Hospital System program. Initially mandated by Congress in 1984 to compare commercial HISs to DHCP, the goal of the Integrated Hospital System program was to test innovative technologies at select testbed sites and migrate them to other VA medical centers operating DHCP. This exposure was intended to allow the VA to identify and test innovations, and then possibly incorporate them into DHCP.

Unfortunately, the Integrated Hospital System program has proved to be costly and largely ineffective. The VA was developing new technologies on top of these four proprietary systems and found it difficult to migrate them to other hospitals. The VA has decided to replace the commercial systems in place with DHCP. A new approach, Hybrid Open System Technology (HOST), is being initiated for integrating innovative commercial technologies into DHCP. The HOST program will replace the four commercial HISs at these hospitals with DHCP, using the DHCP core as the base to test and further develop commercial innovations. The goal of HOST is to integrate commercial software into DHCP under a common user interface. A commercial program will be used in those cases where it is superior to the DHCP offering or where no DHCP product is available.

4.0 VA POSITION ON MUMPS

The VA has been very active in promoting MUMPS and in standardizing the language. It has provided added value for other MUMPS users by providing public-domain DHCP software, developing MUMPS software tools such as FileMan, and funding and performing research to further improve and standardize the language. Because of its substantial investment in MUMPS applications, the VA has assumed a prominent role in the MUMPS community.

The MUMPS community in turn has benefited greatly from the VA's involvement. The VA has continued to develop the MUMPS Language Standard and has been a primary source of funding for MDC activities. No other organization has contributed to and influenced the standardization and advancement of MUMPS more than has the VA. With its involvement in the language, the VA is an excellent source of expertise on the use of MUMPS. Therefore, the VA's evaluation of, future commitment to, and future use of MUMPS within the VA are important considerations in evaluating the MUMPS environment. The VA's position and commitment to the future use of MUMPS will have a significant effect on the future prospects of the language.

4.1 VA Evaluation of MUMPS Language

The VA has an expert knowledge base concerning the MUMPS language, having used it in the agency since 1982. The VA concluded in the mid-1980s that MUMPS is the most cost-effective solution for continued DHCP development. Additional information regarding these cost studies may be obtained from the VA's MIRMO, which coordinates the activities of the ISCs.

Based on survey responses and interviews, the VA considers MUMPS to be an excellent match for DHCP. Although the VA has identified disadvantages related to MUMPS and the MUMPS environment, in its opinion the advantages far outweigh the disadvantages.

4.1.1 ADVANTAGES OF MUMPS

Based on its experience with DHCP, the VA believes that MUMPS generally provides better productivity, maintainability, usability, and portability than other languages. Data base performance and the standardization process also strengthen the MUMPS environment. With a hierarchical data base structure and variable-length records, the VA feels that MUMPS can offer superior data base performance and efficient storage. The portability of MUMPS to various platforms has afforded the VA a great deal of independence between hardware and software. Because of having experienced advantages from MUMPS, the VA is committed to using it for future DHCP development and maintenance.

Over the years the VA has amassed considerable expertise with MUMPS and is able to use this expertise to its advantage. The VA has been able to realize these advantages with MUMPS because the design and primary functions of DHCP are an excellent match with the strengths of the language.

4.1.1.1 Productivity

The VA considers MUMPS to provide a productive development environment for DHCP applications. Based on survey and interview responses, the VA feels that the following features of MUMPS have contributed to improved productivity:

- Exceptional string handling
- Powerful operators and functions
- Interpretive operation
- Rapid prototyping
- Advanced query capabilities through pattern matching
- Integrated data base system

The VA believes that these MUMPS features, as compared to those of other languages, provide productivity advantages for DHCP.

4.1.1.2 Maintainability

The VA feels that MUMPS provides maintenance advantages over other programming languages. MUMPS' method of handling data types greatly facilitates the maintenance of program code for the VA. Changes in the size of a data field or array are handled internally by MUMPS as a result of its variable-length record structure and lack of declarative statements.

The VA also believes that maintenance issues related to MUMPS and most other languages are largely a matter of design and style. Because of MUMPS' flexibility, the VA finds that programmers must be trained for structured programming.

4.1.1.3 Portability

The VA believes that the productive environment of MUMPS and the existence of a portability standard has made it easy to build portable applications. By using the Kernel, the VA has done an excellent job of shielding implementation-specific code, which is likely to be nonportable, from the core DHCP application code. Also, by following the guidance provided in the MUMPS Portability Standard, the VA has avoided the use of nonportable code whenever possible.

Because of the portability of the DHCP code, the VA has been able to achieve independence between the hardware and the DHCP software, minimizing the effort required when changing hardware platforms. The VA continually evaluates new MUMPS hardware platforms to determine their suitability based on a cost-benefit analysis. Thus the VA is able to select the optimum hardware platform with the confidence that DHCP will operate in that environment. The VA has made two major hardware platform changes for DHCP, both of which required no modifications to any of its application code. The first change was to convert some VAMCs from Digital PDP-11s to Digital VAXes. The most recent hardware change was to convert some of the remaining sites with Digital PDP-11s to networked IBM PC-compatible 486-based computers. Individual sites were switched from the PDP-11s to the 486s over a single weekend, with most of the effort spent converting data, not code.

In 5 to 8 years, the VA expects to make another hardware platform change and anticipates no difficulties based on its past experience with MUMPS. The strength of the MUMPS Portability Standard, combined with the modular architecture of the DHCP code, allows the VA to change the underlying hardware as required.

4.1.1.4 Ease of Learning

The VA often hires individuals who are experts in the area that will be automated and then trains them in the MUMPS language, which has a more limited command set to be learned than other languages. The VA believes that DHCP has been improved through teaching clinical experts the MUMPS language rather than teaching programmers clinical applications. With its large staff of programmers with MUMPS expertise, the VA has been able to transfer this knowledge to its clinical experts. The VA feels that MUMPS' ease of learning has provided a significant advantage in DHCP development.

4.1.1.5 Data Base Performance

MUMPS provides fast and efficient data base access, and the VA believes that MUMPS provides the best performance for data base-intensive applications, particularly those applications working with very large (1 gigabyte or more) data bases. The VA also believes that MUMPS does not require the same level of attention to the data base that is required in a DB2, Oracle, or comparable environment. MUMPS implicitly handles many of the performance tuning mechanisms that must be performed explicitly with other systems. The VA looks forward to the SQL binding (1993 ANS) that will extend the capabilities of the language by supporting embedded SQL statements.

The VA considers MUMPS to be superior to Oracle in effectively managing data bases exceeding 1 gigabyte of storage. The planned use of Oracle will be limited to supporting specialized reporting requirements that are more easily met with an RDBMS than with MUMPS.

4.1.1.6 Standardization

The VA has been a catalyst for the MUMPS standardization efforts, providing the funding for much of the ANSI standardization activities of the MDC. The VA sponsored the first NIST MUMPS FIPS PUB (FIPS 125) and is a primary sponsor of the MDC and other standards activities and research. The VA feels that the standardization process and establishment of a portability standard are two of the strengths of the MUMPS environment. The VA will continue to be deeply involved in advancing the language, although it is looking for more active leadership from other MUMPS users.

4.1.2 DISADVANTAGES OF MUMPS

The VA recognizes that there are also disadvantages to the use of MUMPS. Based on its experience with DHCP, the most notable disadvantages do not result from limitations of the language itself, but rather are based on the perceptions other computer professionals hold about the language.

Because of the small number of MUMPS users, the MUMPS community is a close-knit group. Relative to other languages, the population of MUMPS users is much smaller and more concentrated in a few application domains, such as medical. The language itself does not receive much exposure outside of this group or the medical profession. MUMPS' lack of visibility and reputation within the computer industry limits its acceptance as a general-purpose programming language.

4.1.2.1 Visibility

The VA feels that MUMPS is not widely known or used as a general-purpose programming language within the U.S. business community. MUMPS' reputation as a programming language specifically for HISs contributes to its poor visibility. MUMPS' lack of visibility in academic circles is another contributing factor. Very few colleges and universities introduce MUMPS in their computer science curriculums.

In comparison with other languages, the current market for MUMPS is small, with 1991 revenue from MUMPS licenses totaling only \$48.5 million. However, the use of MUMPS as a general-purpose programming language in foreign countries is increasing. In 1991, 45 percent of the MUMPS license revenues was attributed to foreign sales. Asia, Europe, and South America are the major foreign markets for MUMPS.

4.1.2.2 Reputation

The VA feels that while MUMPS has some disadvantages, the language suffers from perceptions acquired from MUMPS' earlier days.

For example, the VA believes that the common perception of MUMPS as a closed system is no longer true. The VA has successfully used windows interfaces, Focus fourth-generation language, and HL7 to access, retrieve, and transfer MUMPS data. Recently the MDC voted to adopt an alternative name for MUMPS, "M", in order to separate the language from its medical origins. The improved support for industry standards planned in the 1993 ANS may help MUMPS shed some of its "closed system" reputation.

Another common perception is that MUMPS has poor maintainability. Based on its experience with DHCP, however, the VA feels that MUMPS' flexibility and interpretive nature provide advantages in maintaining DHCP software. The VA believes that maintainability is much more of a style and design issue than a language issue. The VA recognizes that because of the flexibility MUMPS provides, programmers must be trained to take a structured approach.

4.1.2.3 Actual Disadvantages of MUMPS

The VA recognizes that there are some true disadvantages to the MUMPS language. Because data is stored physically as character strings in hierarchical arrays, MUMPS provides poor performance and support for advanced arithmetic and statistical functions. MUMPS lags behind many RDBMSs in standardized support for transaction processing. Although available in vendor implementations of MUMPS, bindings to SQL, X-Windows, and GKS have not yet been standardized. Generally, the VA feels that its concerns about MUMPS in these areas are addressed adequately in the 1990 ANS and proposed 1993 ANS.

4.2 VA Proposed Enhancements to MUMPS

The VA has been engaged in efforts to extend the capabilities of its MUMPS environment in the following areas:

- Transmission Control Protocol/Internet Protocol (TCP/IP) communications
- Windows-based graphical user interface
- GKS binding
- X-Windows binding
- SQL binding
- Transaction processing support

The VA has begun developing TCP/IP communications for MUMPS systems. A window-based graphical user interface and imaging capabilities for DHCP currently are being developed. The

VA has been active in promoting and sponsoring the proposed GKS and X-Windows bindings for the 1993 ANS. The VA desires support for transaction processing and a binding to SQL in future MUMPS language standards. Both of these are proposed in the 1993 ANS.

The imaging application is of particular interest because of its potential to integrate all information related to a patient record, which allows all records pertaining to a patient case to be reviewed by a physician through DHCP. Dr. Ruth Dayhoff of the Washington ISC and NIST are both involved in the design of this system, which is planned to be piloted at the Baltimore VA hospital.

4.3 VA Commitment to MUMPS and DHCP Development

The VA has stated that MUMPS would still be the language of choice if the agency could choose a new environment for DHCP. The VA has made a long-term commitment to the MUMPS environment for DHCP; the DHCP data base administrator has confirmed that the VA is committed to using MUMPS for DHCP development and maintenance for at least the next 10 years. The VA is satisfied that its concerns related to MUMPS will be addressed in future MUMPS standards and believes that the advantages of using MUMPS for DHCP development and maintenance far outweigh these concerns.

However, while retaining MUMPS for the development of DHCP applications, the VA will continue to evaluate alternative or adjunct environments (such as Oracle and SAS) to MUMPS. Oracle and SAS are used currently to support statistical and ad hoc reporting requirements though data exports from DHCP. Interfacing these environments (such as Oracle and SAS) and other commercial off-the-shelf software (time and attendance, etc.) used within the VA with DHCP will be emphasized in the future. Interfaces to external systems, such as those of the Veterans Benefits Administration, National Cemetery, and DoD, also will be under consideration to improve VA services.

The VA plans to use more tools for MUMPS development, reducing the amount of manual code created. Currently, less than 50 percent of MUMPS code is generated manually. The VA's objective in using development tools is to move the use of MUMPS by programmers to a higher level, with code generators handling much of the routine or mundane programming tasks.

The VA plans to use DHCP's portability to implement new hardware technology based on a cost-benefit ratio. A specific target or "ideal" hardware platform has not been defined by the VA. With the portability that has been built into DHCP, the VA does not have to develop code for a specific platform. Current technology of interest to the VA are client/server architectures and scalable, configurable hardware platforms. Porting DHCP code to different hardware platforms is a relatively simple task, with modifications to hardware-specific code isolated within the Kernel.

4.4 Summary

The VA currently has a 10-year commitment and has invested a minimum of \$800 million in MUMPS applications, and continues to develop MUMPS applications. The VA is an influential participant in setting standards and extensions for the language. The VA also participates in joint development projects with the IHS and DoD. In addition, the VA has made modules of DHCP available to other MUMPS users at a minimal cost to spread the use of the technology. FileMan and other Kernel programs are also available separately as MUMPS development tools.

The VA is dedicated to remaining in the MUMPS environment for the next 10 to 20 years. Based on cost studies performed in the mid-1980s, the VA concluded that MUMPS provides the most cost-effective environment for DHCP, and is committed to the language. However, they will continue to evaluate the capabilities of other data management systems as alternatives and adjuncts to MUMPS. The use of Oracle and SAS will become more prominent in the future, with the objective of integrating these environments within DHCP for specialized uses.

The VA believes that for DHCP, MUMPS has proven to be superior to other languages in productivity, portability, maintainability, ease of learning, and data base performance. Continued use of MUMPS for DHCP is considered cost effective. The VA has made a long-term commitment to the use of MUMPS based on the advantages it has experienced with the language.

5.0 SUMMARY

This section summarizes the use of the MUMPS programming language within the VA and the VA's position on the future use of the language. The objective of the study is to report on the VA's position with respect to the following:

- The features of the MUMPS programming language and proposed extensions to the 1990 ANS
- The use of MUMPS for maintaining existing systems and developing new systems
- The advantages and disadvantages of using MUMPS in a development and maintenance environment

The following sections summarize the results of the study based on discussions with the VA, MUG, MDC, and other MUMPS industry experts.

5.1 MUMPS Programming Language

MUMPS is an interactive programming language and general-purpose DBMS that has become popular for the development of medical information systems. In addition to medical systems, MUMPS supports a variety of other AISs. MUMPS is an approved ANSI, FIPS, and ISO language. MUMPS is best known for its ability to handle unstructured textual information, its data sharing capabilities, and its interpretive nature. There are many features of MUMPS not commonly found in other programming environments.

MUMPS retains characteristics derived from its origins as an interactive text data management system. The most important characteristics are as follows:

- MUMPS is an interpreted language.
- MUMPS supports variable-length records.
- MUMPS supports persistent, shared data.
- Data structures are in the form of sparse, hierarchical arrays.
- Data items are physically stored as character strings.
- Data types and structures do not need to be declared.

The interpretive nature of MUMPS provides for an interactive development environment and facilitates rapid prototyping, maintenance, testing, and debugging. As a programming language, the sparse, hierarchical array structure of MUMPS variables are more adaptable to change than the more common static matrix array structure of variables found in many languages.

However, MUMPS lacks the ability to perform complex statistical or mathematical functions that a matrix array structure provides. MUMPS is devoid of formal declarative statements. Variables are automatically available to all routines running in the same user partition, and constructs must be used to avoid the unintentional reuse of local variables.

The advantages and disadvantages of these characteristics must be weighed carefully. To determine its fitness for use, any evaluation of MUMPS should consider the intended target application. Because of its flexibility, MUMPS requires a structured development environment that enforces conventions and standards for application development and maintenance.

5.2 VA Usage of MUMPS

The VA uses MUMPS to develop and maintain several large AISs operating in VA hospitals and facilities. The VA is the largest user of the MUMPS programming language in the United States, with a programming staff of over 400. These programmers support 168 hospitals and 12 clinics nation wide. The VA has been using MUMPS for enterprise-wide computing since the development of the DHCP in 1982. DHCP automates virtually all of the VA's information tracking and reporting functions and is deployed at 168 of the 172 VA hospitals.

The DHCP program consists of application modules that support various functional areas. The core of DHCP applications is centered around hospital clinical areas, hospital management/administration, and system services.

The DHCP application modules interface with a Kernel, rather than with the underlying hardware and operating system. The Kernel eliminates the need for implementation-specific code in application programs, maximizing the portability of the application code. The VA has instituted software development policies and procedures that allow it to standardize its software both functionally and technically. This further enhances the quality, maintainability, and portability of their software applications. The portability of DHCP has been a significant advantage for the VA in implementing new hardware platforms.

In the future, the VA plans to integrate the use of Oracle and SAS within DHCP and to develop DHCP interfaces to external systems to improve VA services.

5.3 VA Evaluation of MUMPS

The VA believes that for DHCP, MUMPS has provided advantages in productivity, maintainability, portability, and ease of learning over other programming languages. The existence of the portability standard is viewed as a strength of the language. MUMPS also has provided excellent data base performance and efficient use of storage space. Based on cost

studies performed in the mid-1980s, the VA concluded that MUMPS is cost effective for continued DHCP development and maintenance.

The VA believes that many of the publicized disadvantages of MUMPS stem from its reputation in the earlier days of the language. MUMPS historically has had a reputation as a closed system. Improved support in the MUMPS language standard (to be included in industry standards scheduled for 1993) should help to change MUMPS' reputation. MUMPS inherently offers poor support for complex statistical and arithmetic functions. The VA has identified transaction processing and lack of bindings to SQL, X-Windows, and GKS as areas in which they would like to see improvement in the language standard. However, the VA is satisfied that these concerns are adequately addressed in the recent 1990 ANS and proposed 1993 ANS. Other tools (SAS and Oracle) are used by the VA to address statistical reports required in DHCP.

5.4 VA Position on MUMPS

The VA currently has a 10-year commitment and a minimum \$800 million investment in DHCP, and continues to develop applications using the MUMPS language. However, the agency continues to evaluate other data management systems as alternatives and adjuncts to MUMPS. The VA will expand the use of other tools, such as RDBMSs and statistical report writers, to address areas in which MUMPS is deficient.

With the large investment in DHCP, that the VA intends to remain in the MUMPS environment is a logical decision. However, the VA's decision is not based predominantly on its existing investment in DHCP. The VA believes that for DHCP, MUMPS provides cost, productivity, and maintenance advantages over other languages and that the 1982 decision to use MUMPS for DHCP development would still hold true today. The VA has made a long-term commitment to MUMPS and expects the language to be used for DHCP development and maintenance for the next 10 to 20 years.

The commitment that the VA has made to the MUMPS environment for DHCP is analogous to the commitment DoD has made to the Ada language. MUMPS, like Ada for DoD systems, is considered cost effective for DHCP development and maintenance.

APPENDIXES

APPENDIX A
MUMPS SURVEY

MUMPS Survey

1. What is your name, title, telephone number, and facsimile number?
2. What are the current host platforms for your MUMPS based AISs?
3. What version and release of MUMPS is currently being used to maintain and develop your AISs (e.g. VAX DSM v5.0)?
4. What implementation specific extensions of MUMPS (not defined in the ANSI specifications) are widely used in your AISs?
5. How many programmers (in-house and contract) develop and maintain these systems? What percentage of development and maintenance work is performed by your staff and what percentage by contractors?

MUMPS Survey

6. What functional upgrades or other plans have been made with respect to your MUMPS AISs? What other programming environments and languages are you considering for future maintenance and development?
7. What other programming languages interface with your MUMPS systems and could be used for future development?
8. What advantages does MUMPS provide over other languages?
9. What disadvantages does MUMPS have in comparison to other languages?
10. What other programming environments and languages do you have direct experience with or knowledge of?

MUMPS Survey

11. Based on your experience with the environments and languages in 7, how does MUMPS compare to each of them with respect to the following, and why?
 - a. productivity
 - b. maintainability
 - c. usability
 - d. portability
 - e. functionality
12. Based on your experience with software development in MUMPS, what changes would you like to see in the MUMPS programming language to help you to meet future user requirements?
13. Please provide additional comments you may have regarding MUMPS not directly addressed in previous questions.

APPENDIX B
MUMPS SURVEY RESPONSES

APPENDIX C
MUMPS INTERVIEW QUESTIONNAIRE

Interview Questionnaire

SUMMARY

The Office of Technical Integration has requested a study of the MUMPS programming language and DoD automated information systems (AISs) written in MUMPS that support the medical business area. Specifically, this study will gather background data on the MUMPS programming language and how it is used. In support of this objective, the following questionnaire provides a summary of the variety of information needed by the study. A personal interview will be conducted to discuss these items in more detail.

1. What are the current host platforms for your MUMPS based AISs?
2. What version and release of MUMPS is currently being used to maintain and develop your AISs?
3. What implementation specific extensions of MUMPS (not defined in the ANSI specifications) are widely used in your AISs?
4. What functional upgrades or other plans have been made with respect to your MUMPS AISs? What other programming environments and languages are you considering for future maintenance and development?
5. What advantages and disadvantages in meeting user functional requirements does MUMPS provide in comparison with other languages?
6. How many programmers (in-house and contract) develop and maintain these systems? What percentage of development and maintenance work is performed by your staff and what percentage by contractors?
7. Based on your experience with other programming environments and languages, how does MUMPS compare to them with respect to: productivity, maintainability, usability, portability and functionality?
8. Based on your experience with software development in MUMPS, what changes would you propose in future specifications of the MUMPS programming language to improve your ability to meet future user and market requirements?

APPENDIX D
MUMPS INTERVIEW

Date: 20 October 1992

Location: 8403 Colesville Road, Suite 200
Silver Spring, Maryland

Attendees: Javier Abbarran -- VA ISC, DC
Dr. Ruth Dayhoff -- VA ISC, DC
Mike Sinisgalli -- EDS
Carla von Bernowitz -- VRI
Gerry Bragg -- VRI
John Hwang -- VRI

Transcript:

Existing MUMPS users:

- Indian Health Service
- Library of Medicine
- General commercial libraries (MUMPS common platform for vendor software)
- Smithsonian Institute
- Public Health Service
- National Institute on Aging
- Bureau of Prisons
- Navy - Norfolk (use FileMan, renamed package)
- Various State Medical including Oregon, Washington, Florida, and Texas
- South America, Brazil in particular;
- Europe, Finland in particular;
- Asia, Japan in particular;

DHCP programs

DHCP applications can be purchased from the VA or through MUG. Many medical organizations are operating DHCP software. Modifications to modules are coordinated with Salt Lake City ISC to retain integration with other DHCP modules. DHCP required applications are the Kernel and FileMan. Application software modules can be added as needed.

MUMPS versions in use by the VA

- Digital Standard Mumps (DSM for PDP-11's)
- Microconetics MUMPS (486 PC's)

DHCP Portability

DHCP has facilitated the portability of applications to establishing programming conventions and styles. Implementation specific code (found in the kernel) is isolated to a file. Only a switch needs to be changed in order for the proper implementation specific files to be used. Most of these files would revolve around calls to operating system procedures and hardware devices.

VA started testing in 1991 porting from PDP-11's to 486 systems. Birmingham was first test site? VA has been able to perform entire conversion from PDP-11 to 486 based systems over a weekend.

The VA has been directed to integrate data across all DHCP modules to provide a "patient view". IHS has developed a package that performs some of this integration (Patient Care).

DHCP testing procedures:

| | | | |
|--------------------------|--------------------------|-------------|-----------------------|
| Alpha Test (1 site) | Beta Test (3-4 sites) | Approval | ISC's |
| | ---> | ---> Letter | ---> Distribute |
| Internal Verification | External Verification | Written | to Sites ¹ |

¹Each ISC is responsible for 23 sites, except for Chicago which is responsible for 40 sites. The VA is currently split into 4 regions.

MUMPS user groups/community

The MUMPS User Group (MUG) appears to be very strong in fostering the sharing of MUMPS development and ideas between users and enforcing standards.

DHCP imaging project

Dr. Dayhoff is testing an imaging system that ties images (stored in DOS files) with patient/chart/test summaries. Linkage to image files is established through FileMan. Baltimore hospital will implement system, integrating it with Siemens Radiology System.

MUMPS generates standard HL7 messages to integrate.

VA case tools

Code generator - FileMan. Used for quick prototyping.

Code integrity checker - verifies syntax according to VA stds.

Data Modeling - not widely used.

FOCUS can tie into FileMan database (what about others)

Non-MUMPS development

Chicago using Oracle for beneficiary/population summary.

SAS for statistics. Export MUMPS data to SAS,...

MUMPS Advantages

- Very good at integrating if standards followed.
- Very powerful query capabilities (pattern matching, searching)
- Interpretive nature of language supports rapid prototyping, facilitates maintenance, testing, debugging.

MUMPS Disadvantages

- MUMPS is weak in statistical functions. Export data to SAS,...
- Not as popular as other environments, but gaining.
- Need SQL binding that will optimize FileMan searches.

MUMPS & DoD (from VA perspective)

DoD needs to be more active in MDC and MUG. VA solely funded NIST activities.
DoD did not collaborate or provide feedback on CHCS (modifications to DHCP).

MUMPS standard

VA supported upcoming? X-windows binding, Open MUMPS Interconnect (OMI) specifications, GKS binding.

APPENDIX E
ACRONYM DEFINITIONS

| | |
|--------|---|
| AIS | Automated Information System |
| ANS | American National Standard |
| ANSI | American National Standards Institute |
| CDC | Center for Disease Control |
| CHCS | Composite Health Care System |
| DBMS | Data Base Management System |
| DHCP | Decentralized Hospital Computer Program |
| DMRIS | Defense Medical Regulating Information System |
| DoD | Department of Defense |
| DRG | Diagnosis Related Group |
| DSS | Decision Support System |
| FIPS | Federal Information Processing Standard |
| GAO | Government Accounting Office |
| GKS | Graphical Kernel Services |
| GUI | Graphical User Interface |
| HCFA | Health Care Finance Administration |
| HIS | Health Information System |
| HIV | Human Immunodeficiency Virus |
| HOST | Hybrid Open System Technology |
| ICR | Immunology Case Registry |
| IEC | International Electrotechnical Commission |
| IFCAP | Integrated Funds Control, Accounting, and Procurement |
| IG | Office of Inspector General |
| IHS | Indian Health Service |
| IMS | Interim Management Support |
| IRDS | Information Resource Dictionary System |
| ISC | Information Systems Center |
| ISO | International Organization for Standards |
| JCAHO | Joint Commission on Accreditation of Healthcare Organizations |
| MAISRC | Major Automated Information Systems Review Council |
| MAS | Medical Administration Service |
| MCCR | Medical Care Cost Recovery |
| MDC | MUMPS Development Committee |
| MIRMO | Medical Information Resources Management Office |
| MTF | Medical Treatment Facility |
| MUG | MUMPS Users Group |
| MUMPS | Massachusetts General Hospital Utility Multi-Programming System |
| NIST | National Institute of Standards and Technology |
| ODS | Operation Desert Storm |
| OMI | Open MUMPS Interconnect |
| OT&E | Operational Test and Evaluation |
| PAID | Personnel and Accounting Integrated Data |
| PDQ | Physician Data Query |
| PDX | Patient Data Exchange |

| | |
|--------|---|
| RDBMS | Relational Data Base Management System |
| RO | Regional Office |
| SAIC | Science Applications International Corporation |
| SAS | Statistical Analysis System |
| SQL | Structured Query Language |
| TCP/IP | Transmission Control Protocol/Internetnetwork Protocol |
| VA | Department of Veterans Affairs; was Veterans Administration |
| VAMC | VA Medical Center |
| VAX | Virtual Addressing Architecture |
| VBA | Veterans Benefits Administration |
| VHA | Veterans Health Administration |

APPENDIX F
REFERENCES

F.1 MUMPS LANGUAGE STANDARDS AND SPECIFICATIONS

The Complete MUMPS: An Introduction and Reference Manual for the MUMPS Programming Language. John Lewkowicz, Prentice-Hall. 1989.

ABCs of MUMPS. Dr. Richard F. Walters, Digital Press. 1989.

ANSI/MDC X11.1 MUMPS Development Committee.

F.2 MUMPS-BASED AISs

Decentralized Hospital Computer Program (DHCP). Department of Veterans Affairs. August 1991.

The Functional Description (FD) for the Expense Assignment System Version III (EAS III). Electronic Data Systems. 1989.

Medical Expense and Performance Reporting System (MEPRS) and EAS III, Data Base Specification. Defense Medical Systems Support Center. September 1992.

HEALTH AFFAIRS MUMPS SYSTEMS SURVEY REPORT

TABLE OF CONTENTS

| <u>Section</u> | <u>Page</u> |
|---|-------------|
| 1.0 EXECUTIVE SUMMARY | 1-1 |
| 1.1 Study Background | 1-1 |
| 1.2 MUMPS Background | 1-2 |
| 1.3 Study Results | 1-2 |
| 2.0 DATA GATHERING | 2-1 |
| 2.1 Sources Consulted | 2-1 |
| 2.2 DoD MUMPS Surveys | 2-1 |
| 2.3 DoD MUMPS Interviews | 2-2 |
| 3.0 MUMPS LANGUAGE | 3-1 |
| 3.1 MUMPS History | 3-1 |
| 3.1.1 MUMPS Development Committee | 3-1 |
| 3.1.2 ANSI Language Standards | 3-3 |
| 3.2 Distinct Characteristics of MUMPS | 3-7 |
| 3.2.1 MUMPS as a Programming Language | 3-7 |
| 3.2.2 MUMPS as a Database Management System | 3-10 |
| 4.0 AQCESS | 4-1 |
| 4.1 AQCESS Background | 4-1 |
| 4.1.1 Admission and Disposition | 4-2 |
| 4.1.2 Clinical Records | 4-2 |
| 4.1.3 Quality Assurance | 4-2 |
| 4.1.4 Ad Hoc Reporting | 4-2 |
| 4.1.5 Embosser Interface | 4-2 |
| 4.1.6 DEERS Interface | 4-2 |
| 4.1.7 Business Office | 4-2 |
| 4.1.8 Appointing and Scheduling | 4-3 |
| 4.1.9 Emergency Room | 4-3 |
| 4.2 AQCESS Development | 4-3 |
| 4.2.1 Architecture | 4-3 |
| 4.2.2 Data Management | 4-4 |

| | | |
|-------|----------------------------------|-----|
| 4.2.3 | Coding Standards and Development | 4-4 |
| 4.2.4 | Maintenance | 4-5 |
| 4.2.5 | Testing and Validation | 4-5 |
| 4.2.6 | Release and Distribution | 4-6 |
| 4.3 | AQCESS Current Usage | 4-6 |
| 4.4 | AQCESS Planned Usage | 4-6 |
| 4.4.1 | MUMPS Alternatives | 4-6 |
| 4.4.2 | MUMPS Enhancements | 4-6 |
| 5.0 | EAS III | 5-1 |
| 5.1 | EAS III Background | 5-1 |
| 5.1.1 | Radiology | 5-2 |
| 5.1.2 | Personnel | 5-2 |
| 5.1.3 | Pathology | 5-2 |
| 5.1.4 | Respiratory | 5-2 |
| 5.1.5 | Batch Merge | 5-3 |
| 5.1.6 | Data File Maintenance | 5-2 |
| 5.1.7 | EAS III Reports | 5-2 |
| 5.1.8 | System Functions | 5-3 |
| 5.2 | EAS III Development | 5-3 |
| 5.2.1 | Architecture | 5-4 |
| 5.2.2 | Data Management | 5-4 |
| 5.2.3 | Coding Standards and Development | 5-4 |
| 5.2.4 | Maintenance | 5-4 |
| 5.2.5 | Testing and Validation | 5-5 |
| 5.2.6 | Release and Distribution | 5-5 |
| 5.2.7 | Documentation | 5-5 |
| 5.3 | EAS III Current Usage | 5-5 |
| 5.4 | EAS III Planned Usage | 5-6 |
| 5.4.1 | MUMPS Alternatives | 5-6 |
| 5.4.2 | MUMPS Portability | 5-6 |
| 6.0 | CHCS | 6-1 |
| 6.1 | CHCS Background | 6-1 |
| 6.2 | CHCS Program | 6-2 |

| | | |
|--------|---|------|
| 6.2.1 | Order Entry/Results Reporting | 6-2 |
| 6.2.2 | Patient Administration | 6-3 |
| 6.2.3 | Patient Appointment and Scheduling | 6-3 |
| 6.2.4 | Laboratory | 6-3 |
| 6.2.5 | Radiology | 6-3 |
| 6.2.6 | Pharmacy | 6-3 |
| 6.2.7 | Nursing | 6-4 |
| 6.2.8 | Clinical Dietetics | 6-4 |
| 6.2.9 | Quality Assurance | 6-4 |
| 6.2.10 | Mobilization and Mass Casualty Operations | 6-4 |
| 6.3 | CHCS Development | 6-4 |
| 6.3.1 | Architecture | 6-5 |
| 6.3.2 | Data Management | 6-5 |
| 6.3.3 | Coding Standards and Development | 6-5 |
| 6.3.4 | Maintenance | 6-5 |
| 6.3.5 | Testing and Validation | 6-6 |
| 6.3.6 | Release and Distribution | 6-6 |
| 6.4 | CHCS Current Usage | 6-6 |
| 6.5 | CHCS Planned Usage | 6-7 |
| 6.5.1 | MUMPS Alternatives | 6-7 |
| 6.5.2 | MUMPS Enhancements | 6-7 |
| 7.0 | SUMMARY | 7-1 |
| 7.1 | Data Gathering Methods | 7-1 |
| 7.2 | Unique Features of MUMPS | 7-1 |
| 7.3 | DoD MUMPS Usage | 7-2 |
| A.0 | APPENDICES | |
| A.1 | Abbreviations | A1-1 |
| A.2 | References | A2-1 |
| A.3 | MUMPS Surveys | A3-1 |
| A.4 | MUMPS Survey Responses | A4-1 |
| A.5 | MUMPS Interview Questionnaire | A5-1 |
| A.6 | MUMPS Interview | A6-1 |

LIST OF EXHIBITS

| <u>Number</u> | <u>Title</u> | <u>Page</u> |
|---------------|---|-------------|
| 3-1 | Timeline of MUMPS Standardization | 3-2 |
| 3-2 | The MUMPS Standards Process | 3-4 |
| 3-3 | Example of String Subscripts | 3-6 |
| 3-4 | Example of MUMPS Block Structuring | 3-7 |
| 3-5 | Calling External Routines from MUMPS | 3-8 |
| 3-6 | Distinctive Characteristics of MUMPS | 3-11 |
| 3-7 | MUMPS Symbolic Operators | 3-12 |
| 3-8 | One-character Commands in MUMPS | 3-13 |
| 3-9 | The Tree Structure of MUMPS Global Arrays | 3-16 |
| 4-1 | AQCESS Architecture | 4-4 |

1.0 EXECUTIVE SUMMARY

The Office of Technical Integration has requested a study of the Massachusetts General Hospital Utility Multi-Programming System (MUMPS) programming language and Department of Defense (DoD) Automated Information Systems (AISs) written in MUMPS that support the medical business area. DoD uses MUMPS to develop and maintain several large AISs operating in military hospitals and clinics. This report documents DoD's current and planned usage of the MUMPS programming language. Telephone interviews, surveys, and personal interviews with DoD and contractor staff were used to gather information about DoD usage. Other organizations and reference materials were consulted to gather information about the MUMPS language itself.

DoD has a significant investment in MUMPS, using MUMPS to automate major portions of all DoD hospitals and some large DoD clinics. DoD depends on MUMPS for uniform workload reports and hospital inpatient administration at all of their hospitals. DoD is preparing to deploy a MUMPS-based system to automate all military hospitals. The DoD is considering whether to convert from MUMPS to Ada and SQL to comply with Open Systems requirements.

1.1 SURVEY BACKGROUND

DoD is a major user of the MUMPS programming language, having developed and deployed Hospital Information Systems (HISs) using MUMPS as the programming language. This study will provide a survey of DoD's usage of the MUMPS programming language with respect to:

- The use of MUMPS for maintenance of existing systems and new system development
- The advantages and disadvantages in using MUMPS in a development and maintenance environment
- An evaluation of the features of MUMPS and proposed extensions to the 1990 ANSI Language Specification

The information presented was gathered by canvassing experts within the DoD and its contractors on both the current use of the MUMPS programming language, and also planned future usage of the MUMPS programming language. Telephone interviews, surveys, and personal interviews were all used to gather information from DoD and contractor staff about DoD's usage of MUMPS. Additional information about the MUMPS programming language itself was gathered from the MUMPS Users Group

(MUG), the MUMPS Development Committee (MDC), reference materials on MUMPS based AISs, and the MDC's MUMPS Language Standard. A more thorough explanation of the interview questionnaire and survey is contained in chapter 2.0 of this report.

1.2 MUMPS BACKGROUND

The Massachusetts General Hospital Utility Multi-Programming System, commonly referred to as "MUMPS", was initially developed by the Laboratory of Computer Science at Massachusetts General Hospital in 1967. MUMPS is both an interactive programming language and a general purpose database management system that has been widely used in the development of medical information systems.

The MDC publishes both a MUMPS Language Standard and a MUMPS Portability Standard. The MUMPS Language Standard has been approved as an American National Standards Institute (ANSI) Standard, a Federal Information Processing Standard (FIPS), and the International Organization for Standardization (ISO) Standard. The MUMPS Portability Standard allows conforming application code to be truly portable among different MUMPS implementations, operating systems, and hardware platforms.

Although MUMPS was originally developed for minicomputers, MUMPS versions are currently available for mainframes, minicomputers, workstations, and personal computers. MUMPS is best known for its ability to handle unstructured textual information, its data sharing capabilities, and the interpretive nature of the language.

1.3 STUDY RESULTS

DoD has a major investment in MUMPS, using it for AISs that automate some functions in all DoD hospitals and some DoD clinics. DoD depends on MUMPS for automating all of its hospitals through the Automated Quality of Care Evaluation Support System (AQCESS) and other pilot projects. DoD has deployed AQCESS at about 160 sites, which include all military hospitals and the largest military clinics. DoD currently depends on the Expense Assignment System, version III (EAS III), a MUMPS-based system that provides consistent uniform reporting of expenses and manpower, as its only workload management tool. EAS III has been deployed at about 200 sites, and will eventually be deployed at 21 additional sites for a total of 221 sites. DoD is planning to deploy the Composite Health Care System (CHCS), a MUMPS-based HIS, that will replace all of the MUMPS pilot systems. CHCS was deployed to 2 Alpha sites and 11 Beta sites for testing and has been approved by the

DoD Major Automated Information Systems Review Committee (MAISRC) for deployment to all military hospitals pending Congressional approval.

In accordance with the DoD's migration towards an Open Systems Environment, Health Affairs is considering converting EAS III and CHCS, which are still under development, to Ada and SQL. AQCESS, which has already been deployed and is being maintained, has also been considered for conversion to Ada and SQL. Health Affairs believes that conversion of AQCESS would not be cost effective, since the system is to be replaced by CHCS.

DoD and its contractors have identified changes they would like for MUMPS; these changes affect security, standards support, data communications, user interface, and performance. A more secure language would separate the source code from the executable code, preventing a malicious user from accessing the source code. Better support of open standards such as SQL and X-Windows would promote easier integration in an Open Systems Environment. Improved data communications, for import and export of data with other systems, would provide better systems integration and data interchange. An improved user interface would make the applications easier to use. Additional performance-tuning tools would allow the system to run more efficiently.

2.0 DATA GATHERING

This chapter describes how data was gathered about the MUMPS programming language and DoD's usage of it and policy toward it. To gather information for this study, a number of experts in DoD and under contract to DoD were consulted to determine DoD's usage and policy with respect to the MUMPS programming language. Experts recommended by MUG were consulted to gather information about the characteristics of the MUMPS programming language. Face-to-face interviews and telephone interviews were used to gather information.

2.1 SOURCES CONSULTED

Various staff from within DoD, under contract to DoD to develop MUMPS software, and experts on the MUMPS programming language were consulted for this survey. Staff within DoD were consulted about DoD current and planned usage of the MUMPS programming language. Technical consultants under contract to DoD were consulted about how they used the MUMPS programming language for DoD. MUG provided references for MUMPS experts, outside DoD, who were consulted on the MUMPS programming language.

The initial contacts were made through telephone interviews. If follow-up discussion was not planned, a short survey was sent to the expert via facsimile. This survey asked the staff member to provide additional information not covered in the interview and to clarify information about how DoD uses MUMPS. If a follow-up discussion was planned, a set of interview questions was sent so he or she could prepare for the interview.

2.2 DOD MUMPS SURVEYS

Surveys were sent to staff within DoD via facsimile. These surveys contained 13 questions designed to gather information on how MUMPS was implemented and used at DoD. The survey also sought to gather information comparing MUMPS to other programming environments and languages with which the respondent had experience. The MUMPS survey is included in Appendix A.3. Appendix A.4 contains the responses to the surveys. Surveys were sent to Mr. Jim Anchutes of Electronic Data Systems (EDS) and Mr. Al Carter of EDS. Surveys were also sent to Ms. Clarissa Reberkenny of the AQCESS Program Office and Ms. Stephanie Soroka of the EAS III Program Office.

2.3 DOD MUMPS INTERVIEWS

To understand DoD's current and planned usage of the MUMPS programming language, face-to-face interviews and telephone interviews were conducted. A face-to-face interview was conducted with MAJ Tony Barnett of the CHCS Program Office. Prior to the interview, a questionnaire was sent to MAJ Barnett via facsimile to allow him to prepare for the interview. The interview questionnaire is included in Appendix A.5. Appendix A.6 contains a transcript of the interview.

Additional telephone interviews were conducted with DoD and contractor staff to gather information about DoD's current and future plans with respect to MUMPS and also how MUMPS was implemented and used at DoD, respectively. Telephone interviews were conducted with Ms. Nancy Adams of Science Applications International Corporation (SAIC) and Ms. Holly Knauert of National Data Corporation (NDC).

3.0 MUMPS LANGUAGE

The MUMPS programming language is a standard programming language for medical information systems. Forty-four percent of MUMPS users are in the health care industry. Governmental users comprise an additional sixteen percent of the user community. The remaining forty percent are commercial non-health care users. MUMPS is popular in Europe, South America, and Asia as a general purpose programming language. In the United States, MUMPS has not gained widespread use outside of the health care industry. An understanding of the manner in which MUMPS was developed and evolved is important in evaluating its appropriateness for various AISs.

3.1 MUMPS HISTORY

MUMPS was initially developed by the Laboratory of Computer Science of Massachusetts General Hospital in 1967. The objective of this effort was to develop an interactive, multi-user, minicomputer-based system that would support a hospital's special need to store, share, and manipulate text related data. Many of the characteristics of MUMPS were designed to support and manage the varied and sparse data typical of the medical environment. Through the years, MUMPS has evolved into a general purpose hierarchical data base management system that can support a wide variety of applications, such as scheduling, finance, and inventory.

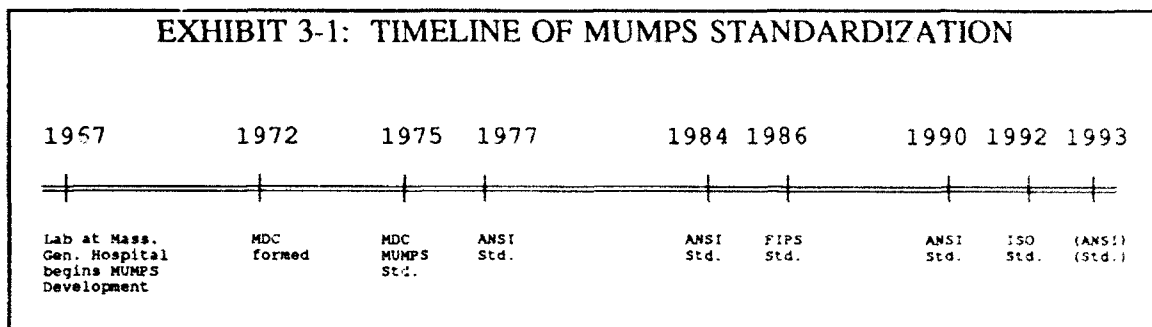
3.1.1 THE MUMPS DEVELOPMENT COMMITTEE

By the early 1970's, several dialects of MUMPS had evolved that were mutually incompatible. The widespread incompatibility severely hampered the portability of MUMPS applications between environments. The MDC was subsequently established to define a MUMPS Language Standard to be submitted to the ANSI. The MDC is composed of subcommittees that address general functional areas of the language, such as user interface or data management; each subcommittee is further organized into task groups which are responsible for a specific aspect of the language, such as X-Windows or string handling. MDC membership is open to any organization that requests membership status and is voted into the MDC; so far, no membership requests have been denied.

Member organization representatives may also join task groups in which they have an interest. The MDC is currently composed of approximately 10 vendor and 50 user organizations. The member organizations send representatives to the three conferences that the MDC holds each year. Much of the conferences are spent working in task

groups. A meeting of the full MDC is held at the conclusion of the conference. The VA has provided a grant to pay for the annual meeting of the MDC; the DoD and Indian Health Service (IHS) have also periodically provided funds in the past. The MDC is also funded through annual membership dues of \$100.

The MDC has continued to assume the responsibility for continued maintenance and development of the MUMPS Language Standard. MUMPS has been approved as an ANSI Standard (ANSI/MDC X11.1-1990), a Federal Information Processing Standard (FIPS PUB 125), and an International Organization for Standardization/International Electrotechnical Commission Standard (ISO/IEC 11756). Exhibit 3-1 outlines the various stages of MUMPS development and standardization.



The MDC has published both a MUMPS Language Standard and also a MUMPS Portability Standard. The MDC is responsible for maintaining the ANSI Language Standard which guides implementors on the minimal requirements that must be met in order to conform to the ANSI Standard. The Portability Standard is a subset of the ANSI Language Standard, serving implementors who create MUMPS interpreters and also developers who create MUMPS applications. Each standard is interpreted as a minimum specification to be met by implementors and a maximum specification to be exercised by developers. If both the implementor and developer adhere to the Portability Standard, the portability of the MUMPS program to other platforms will be assured. The existence of the Portability Standard has been a great benefit to MUMPS users. Both MUMPS developers and end users of MUMPS applications are able to take advantage of advancements in computer hardware and the greater cost/performance ratio associated with newer technology.

Implementations of MUMPS may contain extensions to the ANSI Standard. Some implementation-specific extensions support operating system and hardware-dependent utilities. Other extensions to the current ANSI Standard include official "Type A Extensions" that the MDC has approved for inclusion in future standards.

Anyone can submit proposals for extensions or modifications to the MUMPS Language Standard. The MDC is responsible for reviewing and approving all

proposed extensions or modifications to the MUMPS standard. The MDC makes the ultimate determination as to whether or not the proposal is adopted as an extension or included in the next submission of the MUMPS Language Standard to ANSI.

3.1.2 MUMPS ANSI LANGUAGE STANDARDS

Exhibit 3-2 shows the MDC's formal process for adopting modifications to the MUMPS Language Standard. The MDC is organized into subcommittees. Each subcommittee is responsible for several task groups. Requests received by the MDC for modifications to the language standard are assigned to the appropriate subcommittee. The subcommittee is responsible for assigning the request to an appropriate task group or forming a new task group if an appropriate one does not already exist.

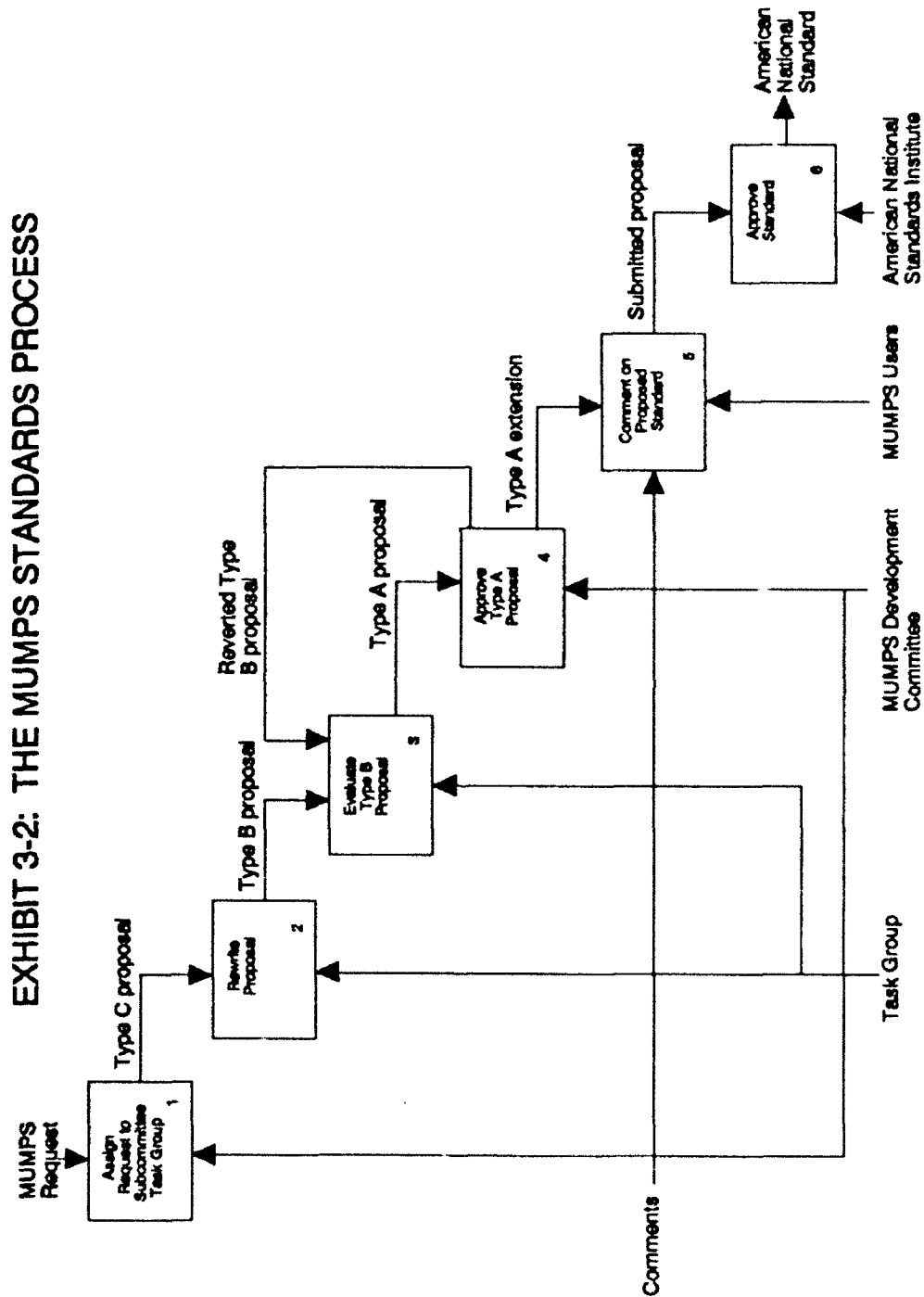
A request at this stage is termed a "Type C proposal." The proposal will be re-written in a formal manner, and if deemed worthy of further consideration it is termed a "Type B proposal." When the proposal is deemed complete enough to be submitted to the full MDC for consideration it becomes a "Type A proposal." When a proposal receives type A status, no other changes to the proposal document will be accepted. If it is determined that changes are necessary, it reverts back to a type B proposal and is returned to the subcommittee for modification.

Type A proposals that MDC approves receive "Type A Extension" status. All Type A Extensions are included in the next submission of the MUMPS Language Standard to ANSI. The MDC will seek comments from the MUMPS community on MDC Type A Extensions to the Language Standard. The comments received are submitted to ANSI along with the MDC MUMPS Language Standard. ANSI will approve the MDC MUMPS Language Standard when it is satisfied with the MDC responses to questions on specific proposals that arose out of the canvassing process. The main role of ANSI in the standardization process is to insure that the MDC has considered the material interests of all affected parties in preparing the MUMPS Language Standard.

The active involvement of the MDC in the continual improvement of the MUMPS Standard has played a significant role in the general portability of MUMPS applications. Proposed extensions to the Language Standard are widely publicized. The MDC actively seeks feedback from MUMPS users on proposed extensions to the ANSI Standard. This collaboration between the standards setting body and the user community has led to a MUMPS Language Standard to which both MUMPS implementors and application developers closely adhere.

Vendor and user consensus over weaknesses in the ANSI Standard are included in future revisions to the standard. Each subsequent revision to the ANSI Standard has addressed specific vendor and user concerns. Changes to the Language Standard that

EXHIBIT 3-2: THE MUMPS STANDARDS PROCESS



are not compatible with previous versions are adopted very cautiously. The most significant events and changes in the MUMPS Language Standard include:

- 1977 Standard -- Merged the several incompatible dialects of MUMPS that had evolved into a single standard
- 1984 Standard -- Implemented the use of descriptive subscripts in MUMPS arrays
- 1990 Standard -- Provided support for structured programming techniques
- 1993 Standard (proposed) -- Will provide support of MUMPS bindings to industry standards (SQL, X-Windows, GKS)

3.1.2.1 1977 ANSI Standard

The 1977 ANSI Standard was based on the initial work of the MDC. The greatest single accomplishment of the MDC was in consolidating the numerous incompatible dialects of MUMPS into a single standard, gaining the consensus of the MUMPS community. The MDC established a formal process of standardizing the language that considered the interests of the MUMPS community. In 1977 ANSI formally verified this process with by approving of the standard.

3.1.2.2 1984 ANSI Standard

The 1984 revision of the ANSI Standard added important extensions to the MUMPS Language Standard. String subscripts were added to the language. This enhancement allowed subscripts to contain descriptive information. The key fields in a MUMPS array provides a description about the data it references. Exhibit 3-3 gives an example of the use of string subscripts in an array. Each subscript represents a different level in the array hierarchy. Multiple subscripts are also allowed. The 1984 revision also included support for the searching of arrays with string subscripts.

3.1.2.3 1990 ANSI Standard

Significant changes in the MUMPS Language Standard were adopted in the 1990 revision. The foremost of these changes were the following additions:

- Structured programming techniques to improve the readability and flow control over MUMPS programs

EXHIBIT 3-3: EXAMPLE OF STRING SUBSCRIPTS

Suppose that the following variables have the values shown:

| <u>variable</u> | <u>value</u> |
|-----------------|--------------|
| PATIENT_SSN | 372-40-1349 |
| DATE | 1/1/90 |
| CLINIC_ID | Radiology |

Suppose further that the user enters "MRI-PATELLA" when the read statement below is executed:

```
READ ! "Enter type of service: ", PROC
SET SERVICE(PATIENT_SSN, DATE, CLINIC_ID)=PROC
```

After these two statements have been executed, the value for service("372-40-1349", "1/1/90", "Radiology") is "MRI-PATELLA".

- Improved formatting and output of numeric data to the language
- Improved functions to query and retrieve data, simplifying programming
- A translation function to change characters in a variable to user specified criteria, such as converting characters from lower to upper case

The MUMPS Standard added the execution of nested blocks of statements. This eliminated the need for non-structured branching in code. Previously label references were required to conditionally execute a set of statements in a program.

Exhibit 3-4 shows two versions of a MUMPS program, one coded according to the 1984 standard, and the second using features of the 1990 standard. Program flow is easier to follow in the second version, since it does not jump back and forth between label references. The support of more structured programming techniques provide a significant improvement in the readability and maintainability of MUMPS programs.

3.1.2.4 1993 ANSI Standard (Proposed Extensions)

Several proposals have been submitted to the MDC for inclusion in the 1993 MUMPS Language Standard. At the time of this writing, the MDC was completing the list of extensions to the MUMPS Language Standard that will be announced to the MUMPS user community and submitted to ANSI. The focus of these proposals addresses

EXHIBIT 3-4: EXAMPLE OF MUMPS BLOCK STRUCTURING

The following MUMPS program conditionally executes a set of statements by referencing labels in the program, in a style typical of code that adhered to the 1984 Standard (everything after a semicolon ";" is a comment):

```
PTN      ;Records patient allergy and ailment information prior to
        ;the performance of any services in a clinic or hospital.
        Read !,"Is the patient allergic to any medication (Y/N)?"
        If ANS["Y" Do MEDCTN
        Do AILMNT
MEDCTN   For N=1:1 Read !,"List medication:",MEDIC Quit:MEDIC=""
        ...Read !,"Describe reaction:",REACT
AILMNT   For N=1:1 Read !,"Describe ailments:",AIL Quit:AIL=""
        ...Read !,"Enter date of ailment:",DATEAIL
        Quit
```

Suppose further that the user enters "MRI-PATELLA" when the read statement below is executed:

```
PTN      ;Records patient allergy and ailment information prior to
        ;the performance of any services in a clinic or hospital.
        Read !,"Is the patient allergic to any medication (Y/N)?"
        If ANS["Y" Do
        .   For N=1:1 Read !,"List medication:",MEDIC Quit:MEDIC=""
        .   ...Read !,"Describe reaction:",REACT
        For N=1:1 Read !,"Describe ailments:",AIL Quit:AIL=""
        ...Read !,"Enter date of ailment:",DATEAIL
        Quit
```

previous shortcomings of MUMPS in the support of industry standard protocols.

MUMPS was originally developed as a closed system. Only MUMPS could access data in its globals, which is the MUMPS term for data files. MUMPS lacked a standard definition of facilities to communicate with other languages and database systems. Because the MUMPS Language Standard did not support these capabilities, vendors added implementation-specific extensions to MUMPS to support these needs. The most noteworthy extensions currently under consideration by MDC include the following:

- External routine calling
- MUMPS bindings to SQL2, X-Windows and GKS
- Standards for Open MUMPS Interconnect (OMI)
- Transaction processing standards

The MDC plans support for each of these elements in the 1993 Language Standard.

External routine calling

In MUMPS, implementation-specific extensions begin with the characters "\$Z". Currently, external routine calling in a MUMPS program is accomplished with \$ZCALL or other \$Z functions. These non-standard functions are vendor-specific and are not portable. Direct support for external callable routines was proposed for inclusion in the 1993 MUMPS Language Standard. External callable routines can return a value to the MUMPS program or simply perform a function external to the MUMPS program. Exhibit 3-5 shows the syntax for calling external routines that has been proposed for the 1993 standard.

EXHIBIT 3-5: CALLING EXTERNAL ROUTINES FROM MUMPS

The MUMPS syntax for calling external routines consists of four elements:

- An ampersand (&) to indicate that the routine or program is external to MUMPS
- The name of the package that provides a binding between MUMPS and the external routine (to be assigned by MDC or implementors)
- A period (.) to separate the package name from the routine
- The name of the external routine or program, including parameters

The following is an example of how a program might call an external routine in an X-Windows system:

```
Set Display=$&XLIB>.OpenDisplay(MYNODE)
```

This method of external routine calling allows MUMPS to integrate with bindings to other languages, interfaces, and other external program libraries. The only portion in the above statement that needs to be defined within MUMPS is the package name "\$&XLIB". XLIB is the name of the package that contains libraries of standard routines with X-Windows and defines the external routine structure.

MUMPS bindings

The external routine calling capability will allow MUMPS to access libraries of functions and bindings to other libraries. Proposed bindings include:

- SQL-2
- X-Windows
- GKS

Future bindings could be added to support POSIX operating systems or other services.

SQL-2 bindings have been proposed to allow SQL statements to be embedded within a MUMPS program. The SQL binding will allow the retrieval and manipulation of data in a MUMPS hierarchical array through both embedded SQL query statements and MUMPS routine calls from within an SQL program. The MUMPS interpreter will

accept interactive and embedded SQL commands from users, allowing the MUMPS system to function as a SQL server. Although several vendors have integrated MUMPS/SQL products, each such integration is vendor-specific and not portable to other vendor products.

An X-Windows binding has been proposed to allow MUMPS to present a standard Graphical User Interface (GUI). MUMPS currently lacks support for the windows, dialog boxes, and menu bars that an X-Windows binding would provide. The binding between MUMPS and X-Windows is defined at the standard libraries level. The MUMPS package name will reference a library of X-Windows routines.

A Graphical Kernel System (GKS) binding has been proposed to address the graphical limitations of the MUMPS language. Currently, MUMPS only supports text output; the GKS binding would add 2-dimensional graphics to MUMPS. In addition, GKS is an open systems technology. The VA has been very active in the GKS binding by providing funding for the testing and development of the binding at the University of Lowell's Graphics Research Laboratory.

Standards for Open MUMPS Interconnect (OMI)

The Open MUMPS Interconnect (OMI) standard will allow implementations of MUMPS written by different vendors to share data in a standardized manner. OMI makes it possible for MUMPS processors to share data among themselves, distributing data throughout a system instead of in a single centralized data repository. This will replace the proprietary data sharing methods that some vendors currently use.

Transaction Processing Standards

The ability of MUMPS to support transaction processing adequately is important for its acceptance as a general purpose business language. There are four elements critical to a transaction processing system:

- Atomicity -- data base updates must occur at the transaction (unit) level; partial updates upon error conditions are not acceptable.
- Consistency -- transactions must result in a consistent data base at all times.
- Isolation -- transactions must be isolated from each other; transactions must produce the same result whether running concurrently or sequentially.
- Durability -- Once a transaction has been reflected in the data base, it must continue in that state regardless of system failures.

3.2 DISTINCTIVE CHARACTERISTICS OF MUMPS

MUMPS has several characteristics that differentiate it from other programming languages. The characteristics of MUMPS should be evaluated according to the intended application of the AIS. Exhibit 3-6 contains a description of the distinctive characteristics of MUMPS. The following sections discuss implications of each characteristic.

3.2.1 MUMPS AS A PROGRAMMING LANGUAGE

Many of the programming characteristics of MUMPS are distinct from those found in other languages. MUMPS did not evolve out of the mainframe environment. The target platform of the language was shared minicomputers operating in an interactive environment. The following sections describe the characteristics and implications of MUMPS as a programming language.

3.2.1.1 Interpretive Nature of MUMPS

The interpretive nature of MUMPS comes from the fact MUMPS programs and source code are interpreted at run time. Source code instructions are interpreted into machine level instructions at the time of execution, rather than being compiled prior to execution.

Debugging is facilitated since the programmer can interact with the program at run-time and execute commands directly at the keyboard. The interpretive nature facilitates rapid prototyping, testing, debugging, and maintenance.

EXHIBIT 3-6: DISTINCTIVE CHARACTERISTICS OF MUMPS

- MUMPS is an interpreted language.
- Multiple, abbreviated statements can be included on a single line of program code.
- Operator precedence in arithmetic expressions is ignored when parenthetical notations are not used, defaulting to strict left to right evaluation.
- MUMPS supports a persistent, shared data base.
- MUMPS variables are physically stored as variable-length character strings.
- MUMPS does not support declaration statements.
- Arrays are heirarchial and require no pre-allocation of memory.
- Data on disk are also stored in a sparse, hierarchial array.
- MUMPS allows the use of string subscripts as keys to array nodes.
- Temporary variable values are not removed from memory upon routine termination.
- Shared data are available to other routines without program linkage or data declarations.
- Data are physically stored in sorted order according to subscript values.

Because MUMPS is an interpreted language, it is less efficient than a compiled language. In order to execute each command, MUMPS must first translate the source code instructions into machine instructions and then execute those machine instructions. Compiled programming languages perform this translation in advance and store the machine level instructions as a program file that is executed at run-time. Compiling programs improves efficiency since the translation of each statement occurs only once, not each time it is operated; in addition, compilers optimize the instructions to perform faster execution.

In order to reduce the overhead associated with run-time interpretation, most implementations of MUMPS "tokenize" the code by partially compiling and optimizing the source code to improve execution speed. Tokenized MUMPS code still requires the MUMPS interpreter to operate. The dynamic structure of MUMPS at runtime makes it impossible to predict the absolute memory addresses, required

variable space, and instructions to be executed; however, this information would be necessary to produce fully compiled MUMPS code.

3.2.1.2 Language Syntax

The syntax of MUMPS programs is typically terse and cryptic. MUMPS makes extensive use of symbolic operators, as shown in exhibit 3-7.

EXHIBIT 3-7: MUMPS SYMBOLIC OPERATORS

The line of code shown below uses the indicated operators to tell MUMPS to check the variable SPONSORID for a sequence of 3 numeric digits, and if it finds them, to write the variable value on column 20, skip a line, with the variable inside the message "ID [sponsorid]: is valid.":

```
If SPONSORID?3N Write !,?20,"ID ",SPONSORID,": is valid."
```

The operators and their meanings are as follows:

| <u>operator</u> | <u>meaning</u> |
|-----------------|-------------------------------|
| ?3N | look for three numeric digits |
| ! | skip a line |
| ?20 | tab to column 20 |

MUMPS has a line orientation whereby several commands can be entered and executed from the same line of source code. Each line in a MUMPS program is limited to 255 characters. To support page printing, screen and display width limitations, continuation of a line is notated by three periods at the start of the next line. Any MUMPS command can be abbreviated to the first letter of the command. Exhibit 3-8 is an example of MUMPS code using one-character commands, two or more to a line.

Combining multiple, abbreviated statements on a line makes code easier to write but harder to read. Programming conventions and standards should be established to enhance the maintainability of the code through adequate program documentation and proper use of MUMPS syntax.

3.2.1.3 Routine Structure

The Portability Standard limits the collective size of MUMPS routines and local variables executing in main memory to 4,000 bytes. The actual allowable routine size

EXHIBIT 3-8: ONE-CHARACTER COMMANDS IN MUMPS

The following MUMPS code fragment creates an array of numbers from one to 10, searches the array, and write the highest and lowest values encountered in the array:

```
For A=1:1:10 Set ARY(A)=A
S HI=0, LO=999999
F A=1:1:10 S:ARY(A)<LO LO=ARY(A) S:ARY(A)>HI HI=ARY(A)
W !,"Highest value = ",HI,!,"Lowest value = ",LO
```

Note the use of one-character command abbreviations:

S for Set

F for For

W for Write

is usually implementation specific. The allowable size of a routine depends on the operating system and hardware issues such as available memory, buffers, disk caches and other jobs. MUMPS reads routines from disk or disk buffers when they are called to execute. Routines are removed from memory upon termination. Routines are swapped between main memory and disk storage as needed. The temporary variable space remains unchanged, allowing other routines access to temporary variables. Program linkage declarations are not supported since routines in a program are not loaded in advance of execution.

The MUMPS routine structure has several implications for the developer:

- Local variables are available for other routines, eliminating the requirement for explicit declarations on the variables.
- MUMPS does not automatically shield the unintentional re-use of locally used variables from the programmer.
- Local variables must be explicitly reinitialized by the programmer to prevent unintentional re-use of local variables.

3.2.1.4 Declarative Statements

Declarative statements for variables, constants, procedures or functions are not supported in the MUMPS language. The following characteristics contribute to the lack of declarative statements in MUMPS:

- MUMPS is a command line oriented language. Declarations are defined at the time the function is required.

- MUMPS does not run in compiled mode so linkages between separately compiled modules are not necessary
- MUMPS variables and arrays are automatically available to all routines operating in the current process.
- MUMPS variables are physically stored as variable-length character strings.

The elimination of declarative statements in MUMPS is a major departure from other languages. Although the elimination of declarations may provide some advantages in writeability, the use of declarations is useful in more clearly defining the roles of variables and procedures in a program.

3.2.2 MUMPS AS A DATABASE MANAGEMENT SYSTEM

MUMPS was originally designed as both a general purpose programming language and a data base management system. The principal function of a Medical Information System is to store, retrieve, and manipulate shared, textual data. Standard MUMPS does not have built-in functions for complex mathematical or statistical applications. Because of its integrated data base system the manner in which MUMPS stores and manipulates data is different from other programming languages.

3.2.2.1 File Structures

The MUMPS file structure is a sparse, hierarchical array implemented in a tree structure. MUMPS arrays are persistent; that is, they exist even after the routine that created them has stopped running. Local variables are shared by all applications using the same directory. MUMPS provides constructs to help the programmer prevent the unintentional re-use of local variables.

Most programming languages implement a sequential file structure in the form of a matrix. The matrix structure provides advantages in performing advanced statistical and mathematical functions. These file structures are usually static and hard to adapt to changing requirements. Selecting individual records from a MUMPS hierarchical file is very efficient compared to sequential files.

3.2.2.2 Data Types

Most DBMSs support some form of basic data types such as integer, character and abstract data types such as date, time, and currency. MUMPS does not support any data type declarations. All data are physically stored as variable length character strings. MUMPS automatically evaluates each variable according to the operation

being performed. A numeric operation will return a numeric data type and a character operation, such as a concatenation, will return a character string type. MUMPS will evaluate a variable as either a character string, integer, floating point number, or Boolean value depending on how it is to be used.

Because numbers are handled as text strings, they must be converted into a more efficient numerical form that the computer can operate on directly. These conversions would not be necessary for a language that supports integer and floating point numbers. Thus MUMPS is much less efficient at complex numerical calculations than a language that does not require the additional overhead of converting strings to machine numbers. In addition, MUMPS requires additional space to store numbers since the packed machine level representations are not available.

MUMPS is essentially a non-typed language. A programmer has an unlimited set of operations that may be performed on data. However, data typing is useful in describing the nature of an element, domains of acceptable values for the element, and logical operations that can be performed on the element.

3.2.2.3 Mumps Arrays

Arrays (referred to in MUMPS as "Globals") have no predetermined fixed or allocated size. Records in an array are variable length. Variable names may have up to thirty-one characters, allowing the programmer to describe the data completely; however, MUMPS uses only the first eight characters to distinguish between variables.

Most languages implement a matrix array structure with a predefined number of columns and rows. The variable length record structure of a MUMPS sparse hierarchical array makes efficient use of resources. Disk storage requirements are determined by the amount of data physically stored, not by a pre-defined array size and record length; null values and redundant trailing spaces are not saved.

The branches in the hierarchical array structure are termed subscripts. Subscripts are the keys to the data contained in each node in the array. Each subscript is a branch in the array, representing a level in the array hierarchy. The physical ordering of data in an array is determined by the subscript values. Instead of storing data in the order they are entered, MUMPS stores data in sorted order according to the subscript values. Subscript values may be integers or character strings of up to thirty-one characters.

Storing in order by subscript values improves the search and retrieval ability of MUMPS. For example, a scan of records with the last name starting with the letter "G" will begin sequentially with the first record beginning with "G", terminating when a name beginning with the letter "H" is encountered.

MUMPS arrays and subscripts are defined at the time a function uses the array. Exhibit 3-9 illustrates the hierarchical array structure of MUMPS.

EXHIBIT 3-9. THE TREE STRUCTURE OF MUMPS GLOBAL ARRAYS

| <u>ARRAY</u> | <u>FIRST SUBSCRIPT (SSN)</u> | <u>FIRST SUBSCRIPT (DATE)</u> | <u>VALUE</u> |
|---------------|--------------------------------------|---------------------------------------|---------------------|
| PATREC | | | |
| | 372-40-1234 | | |
| | | | |
| | 372-40-1349 | 12/1/91 | "John Smith" |
| | | 1/1/92 | "Ultrasound" |
| | | 1/23/92 | |
| | | | |
| | 372-40-2299 | | |
| | | | |

The following statements assign the values shown in the boxes indicated in bold above:

```
Set *PATREC(372-40-1349)="John Smith"
Set *PATREC(372-40-1349,010192)="Ultrasound"
```

The MUMPS hierarchical array structure preceded the relational table model. As in the relational model used in Relational Data Base Management Systems (RDBMSs), the data are normalized and redundant data are not stored within the system. Tables and attributes in RDBMSs are explicitly defined in advance of their use; in MUMPS, the "tables" and attributes are implicitly defined when they are used. The MUMPS sparse hierarchical structure is most useful for storing textual or other data whose records are of varying length. User views of the data in a RDBMS must be explicitly defined through the relationships between table values; however, hierarchical data bases implicitly contain relationship information that one record is dependent upon another.

Unlike the relational model, entity relationships are not joined by matching values in key fields; instead, they expressed through the hierarchy of the tree. This has the

following implications:

- A relational data base is more flexible than a hierarchical data base, since tables can be related to each other in arbitrary ways, as opposed to only through dependency
- A hierarchical data base can be much faster to use, since the records are implicitly related, and two sets of data do not have to be inspected and compared to locate matching values for each join

Different user views of a MUMPS array would require the view to be defined at the physical level, storing an additional array with redundant subscripts and nodes. The inability to support different user views from the same physical structure is a significant drawback. If many user views are necessary, the storage efficiency of variable length record structures is largely negated by additional redundant data.

4.0 AQCESS

The Automated Quality of Care Evaluation Support System (AQCESS) collects and reports clinical, administrative, and managerial information to support inpatient administration of DoD's medical quality assurance programs. AQCESS is used by 160 military hospitals and large military clinics. It can run on three different hardware architectures.

DoD selected MUMPS as the programming language so that AQCESS could reuse software developed by the Department of Veterans Affairs (VA). Because AQCESS is being replaced by the CHCS, DoD does not plan to convert it to Ada and SQL.

4.1 AQCESS BACKGROUND

In 1984, the Assistant Secretary of Defense (Health Affairs) identified medical quality assurance as a priority automation requirement within the military health system. The Tri-Service Medical Information Systems Program Office, in conjunction with functional and technical representatives from each military department, was requested by the Deputy Assistant Secretary of Defense (Professional Affairs and Quality Assurance) to develop and deploy a microcomputer-based information system at all DoD hospitals by the end of 1985. DoD originally awarded the AQCESS software development contract to National Data Corporation (NDC). Later, maintenance for AQCESS was included under the Electronic Data Systems (EDS) contract for Systems Integration, Design, Development, Operations, and Maintenance (SIDDOMS). EDS subcontracted part of the responsibility for maintaining AQCESS to NDC.

AQCESS was developed in modules that were implemented in phases. The early phases of deployment included:

- Admission and Disposition
- Clinical Records
- Quality Assurance
- Ad Hoc Reporting
- Embosser Interface

Later, the following additional modules were deployed:

- DEERS Interface
- Business Office
- Appointing and Scheduling
- Emergency Room

4.1.1 ADMISSION AND DISPOSITION

The Admission and Disposition module addresses patient registration, admission, transfer, and disposition. It also addresses bed management, inpatient history, and reporting.

4.1.2 CLINICAL RECORDS

The Clinical Records module records documentation on patient episodes and diagnoses. It also supports procedure data capture, patient day computation, record tracking, and reporting.

4.1.3 QUALITY ASSURANCE

The Quality Assurance module supports inpatient and emergency room occurrence screening, problem tracking, provider profiling, credentialing, incident reporting, blood utilization review, and reporting.

4.1.4 AD HOC REPORTING

The Ad Hoc Reporting module allows users to produce AQCESS reports that meet the MTF's immediate and continuing special needs.

4.1.5 EMBOSSER INTERFACE

The Embosser Interface module supports embossers from Addressograph, National Business Systems, and Datacard.

4.1.6 DEERS INTERFACE

The DEERS Interface provides the ability to check a patient's eligibility at the time of admission against the Defense Enrollment Eligibility Reporting System (DEERS).

4.1.7 BUSINESS OFFICE

The Business Office module lets users calculate inpatient bills, post payments received, and issue receipts and invoices.

4.1.8 APPOINTING AND SCHEDULING

The Appointing and Scheduling Module supports the creation and maintenance of clinical provider schedules; appointing, scanning, and canceling patients; recording patient encounters; and producing management reports.

4.1.9 EMERGENCY ROOM

The Emergency Room module allows users to record patient emergency room visits; produces an automated SF Form 558, Emergency Treatment Record; produces the automated emergency room log; and stores a history of the encounter on-line.

4.2 AQCESS DEVELOPMENT

Because MUMPS does not support typed variables and lacks a number of structured programming constructs, EDS and NDC had to institute conventions and standards to manage the software development process. EDS found that MUMPS code was not difficult to maintain as long as the software had adequate controls upon it, and the software was written in a structured way.

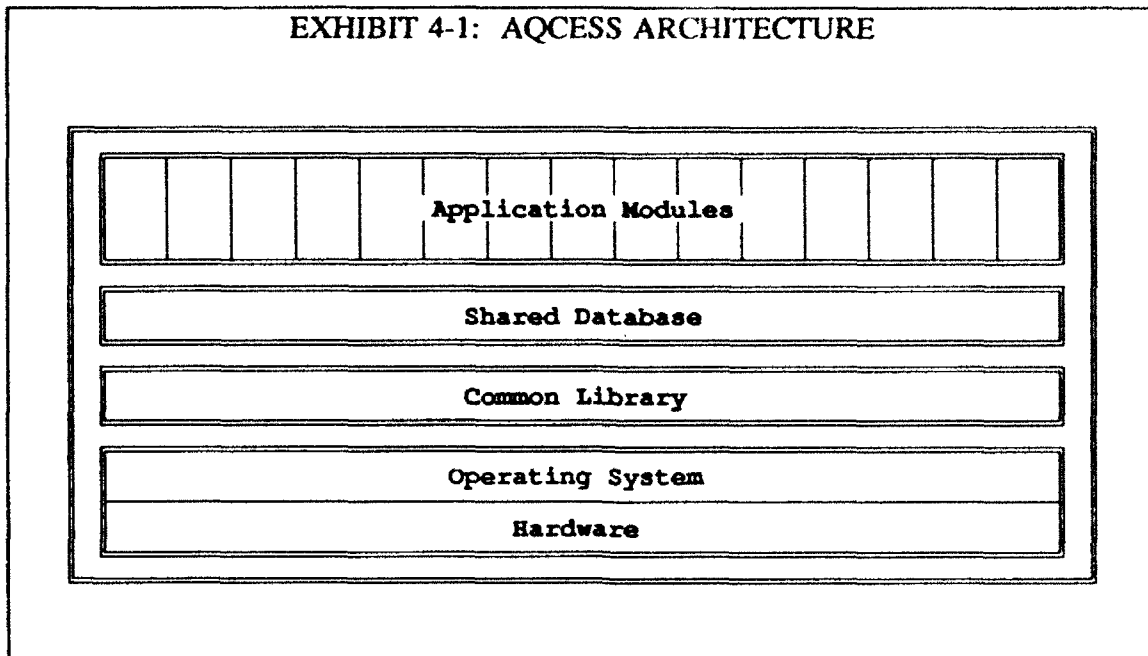
AQCESS was developed in MUMPS to take advantage of and reuse the software that the VA had already developed for DHCP. AQCESS is being developed on 486-based Unix PCs running Micronetics Standard MUMPS (MSM).

4.2.1 ARCHITECTURE

AQCESS was designed with a layered architecture to insulate the application code modules from the underlying hardware and operating system. The user interacts with an individual application module which accesses a library of shared routines. These libraries provide general hardware support to insulate application code from the underlying hardware and operating system. Exhibit 4-1 depicts the modular architecture of AQCESS.

The layered design uses the common libraries to replace the functions of the operating system. EDS and NDC use sets of common library routines for all data input and output, patient lookups, function key handling, and on-line help documentation. AQCESS application software is portable to any system that the common library routines can be ported to. The modular design allows AQCESS to be expanded with additional application modules as they are available or needed. The modular design facilitated the phased deployment by allowing the initial modules to be augmented later.

EXHIBIT 4-1: AQCESS ARCHITECTURE



4.2.2 DATA MANAGEMENT

The VA developed a data manager called FileMan to provide standard file and data management services for DHCP using an active data dictionary. EDS and NDC use FileMan-compatible active data dictionaries to manage data. EDS has ensured compatibility with FileMan by developing a set of tools that use the FileMan structure. EDS also uses FileMan for data base design. Data elements follow naming conventions which were established by EDS.

4.2.3 CODING STANDARDS AND DEVELOPMENT

To manage the development of MUMPS software and improve the maintainability of the code, EDS and NDC use Programming Support Environments and CASE tools. EDS and NDC established coding standards and conventions to standardize the code and make it easier to maintain.

EDS has developed a Programming Support Environment in 'C' to support MUMPS development and maintenance. EDS also uses CASE tools to generate about 60-65% of the code; these code generators are used to create and maintain the code for display screens and printed reports.

EDS has code standards and style guides and employs automatic syntax checkers to verify code. Naming conventions for routines were also established by EDS. In addition, during code review, the reviewers check the syntax for conformance to standards.

NDC developed a proprietary Programming Support Environment in MUMPS to support their MUMPS programming. NDC used CASE tools to generate about 40% of the code, automating display screen and report generation. Other tools created and maintained on-line documentation, edit criteria tables, and system queries. NDC also used a tutorial generator to create on-line tutorials for users.

4.2.4 MAINTENANCE

EDS has instituted a formal change process to maintain AQCESS software. Each "service report" identifying an error or "service change request" suggesting a change received by the maintenance staff is assigned to a developer. The developer determines the scope of the change needed to implement the service report or service change request. The developer then checks the code out from EDS's Code Management System (CMS) and makes the required changes. The changes to the MUMPS code are reviewed, and the functionality of the code is tested. If the changed code is approved, then the developer checks it back into the CMS, where it becomes part of the new baseline from which new AQCESS releases will be made.

Because AQCESS is written in MUMPS, EDS finds AQCESS software is easier to maintain than if it were in Common Business-Oriented Language (COBOL). Testing and debugging AQCESS code is facilitated by the availability of all variables and the line of source code that caused the error, features typical of an interpreted language like MUMPS.

4.2.5 TESTING AND VERIFICATION

Three levels of testing and verification are performed upon AQCESS software before it is released. First, each MUMPS routine is unit tested individually. Then the routines are integration-tested to verify that they interact together properly. Finally, release verification is performed during the release and distribution of software.

4.2.6 RELEASE AND DISTRIBUTION

EDS has configuration management controls on the AQCESS software that is to be released. AQCESS software releases are distributed and loaded from magnetic tape. Prior to release and distribution, EDS makes a test tape, and verifies that the new

AQCESS software will load and run on all three target platforms: Digital Equipment Corporation's (DEC) PDP minicomputers, DEC VAX superminicomputers, and 486-based personal computers (PCs).

4.3 AQCESS CURRENT USAGE

AQCESS is currently installed at all military bases with hospitals. AQCESS is also installed at all military clinics that are large enough to use it. Due to base closures, AQCESS is now installed at only about 160 sites, down from the 168 sites at which it was previously installed. Most installations run AQCESS on a DEC VAX computer running Digital Standard MUMPS (DSM). Although DSM running on a VAX supports 'C' calls to and from MUMPS, AQCESS does not use them.

4.4 AQCESS PLANNED USAGE

AQCESS is scheduled to be replaced by CHCS as CHCS is deployed at military hospitals. Therefore DoD does not plan to convert it to Ada and SQL. EDS and NDC have identified standard data interface, user interface, and Open Systems Environment areas in which MUMPS could be improved.

4.4.1 MUMPS ALTERNATIVES

The AQCESS Program Office is not seeking alternatives to MUMPS for continued maintenance or development, since the system is scheduled to be replaced by CHCS over the next decade. A complete reengineering effort to convert AQCESS from MUMPS to another language is neither feasible nor cost effective. AQCESS will have some new functionality added to it since CHCS will be deployed in phases over five years; many sites will still use AQCESS until CHCS is fully deployed. The Program Office does not expect to make major changes to AQCESS, however.

4.4.2 MUMPS ENHANCEMENTS

Both EDS and NDC believe that MUMPS should support better interfaces to other DoD systems, such as DEERS, with a more open architecture. NDC believes that the MUMPS language should be more structured. NDC would like MUMPS to support Graphical User Interfaces (GUIs) to present a more user-friendly interface. EDS also believes that MUMPS should have better interfaces for data transfers. Standard interfaces between systems would facilitate many of the data transfer functions and interfaces, such as interfacing with DEERS, exchanging files with PCs, and interfacing with applications

such as an encoder/grouper system for diagnosis related groups (DRGs).

5.0 EAS III

The Expense Assignment System, Version III, (EAS III) is DoD's only medical workload management system. It is installed at all DoD hospitals. It consolidates information gathered from the various financial and personnel systems at military Medical Treatment Facilities (MTFs) and generates uniform expense and performance data. EAS III runs under the Unix operating system on two different hardware architectures.

DoD selected MUMPS as the programming language to ensure compatibility among EAS III, AQCESS, and CHCS. DoD plans to consider converting EAS III to Ada and SQL at some point in the future in order to comply with DoD's open systems environment.

5.1 EAS III BACKGROUND

Version 1 of EAS was developed to consolidate the financial information of military MTFs into consistent, uniform categories for expense and manpower reporting. In 1973, a DoD study indicated that there were no uniform financial record keeping practices between the service branches. Therefore, a Uniform Chart of Accounts workgroup was established to develop a Uniform Chart of Accounts (UCA) for financial reporting. Concurrently, a Uniform Staffing Methodology (USM) workgroup was established to identify the hospital personnel categories within MTFs. In 1985, Health Affairs merged the reporting requirements of UCA and USM with EAS processing into the Medical Expenses and Performance Reporting System.

EAS III, introduced in 1990, was developed and is being maintained by EDS. MTFs prepare financial information on a monthly cycle for processing on the EAS III system. EAS III receives data from the following service branch financial systems:

- Standard Financial System (STANFINS)
- Navy Standard Claimant Accounting Module (NSCAM)
- Uniform Medical Expense Reporting System (UMERS)

STANFINS is the Army's financial system. NSCAM is the financial system of the Navy and Marine Corps. UMERS is the financial system of the Air Force. EAS III also receives data from service branch personnel systems and the EAS III Personnel subsystem.

EAS III formats, stores, and processes the expense and performance data transmitted from service branches to produce files and reports for submission to Health Affairs. The primary function of earlier versions of EAS was the remote processing of files

transmitted from MTFs. Enhancements included in EAS III allow it to support financial reporting, workload management, and planning at the local MTF level. EAS III comprises the following subsystems:

- Radiology
- Personnel
- Pathology
- Respiratory
- Batch Merge
- Data File Maintenance
- Processing
- Reporting
- System Functions

5.1.1 RADIOLOGY

The Radiology subsystem logs radiology patients and workload, produces daily and monthly logging reports, and generates radiology reports.

5.1.2 PERSONNEL

The Personnel subsystem maintains the Master Personnel File for the purpose of reporting hours for MTF personnel and producing Full Time Equivalents (FTEs) and salary expenses for EAS III processing.

5.1.3 PATHOLOGY

The Pathology subsystem logs daily pathology transactions, and produces daily and monthly logging reports. At the end of each month, the Pathology Subsystem workload data is merged with EAS III workload data.

5.1.4 RESPIRATORY

The Respiratory subsystem logs respiratory therapy workload and produces daily and monthly logging reports for the respiratory therapy department.

5.1.5 BATCH MERGE

The Batch Merge subsystem supports external system interfaces to Service Financial and Service Personnel systems. This subsystem accepts data in Source Data Collection (SDC) or Service Unique Interface (SUI) formats and edits them against the EAS III database.

5.1.6 DATA FILE MAINTENANCE

The Data File Maintenance system provides for updating, printing, and viewing EAS III data files.

5.1.7 PROCESSING

The Processing subsystem supports the core of the EAS III program. The Processing subsystem performs validations, quality control, calculations, and edits on data entered by batch merge and data file update functions.

5.1.8 EAS III REPORTS

The EAS III Reports system allows EAS III users to print pre-defined summary reports, audit trails, and error reports, and to view or print files prepared by Health Affairs.

5.1.9 SYSTEM FUNCTIONS

The System Functions subsystem transmits EAS III prepared files for the DoD, performs data backups and restorations, maintains master file tables, accepts electronic transfer of software updates, and generates system security reports.

5.2 EAS III DEVELOPMENT

Health Affairs required the use of MUMPS as the development language for EAS III to ensure compatibility with CHCS and AQCESS. Earlier versions of EAS were developed in COBOL and operated on a centrally located mainframe.

5.2.1 ARCHITECTURE

EAS III was designed with a layered modular architecture to insulate the application code modules from the underlying hardware and operating system. The user interacts with a single module which uses an interface code library. The interface code library provides generates different user interfaces, general hardware support, and performs input-output functions. Depending upon the user, the program will select the appropriate code for the appropriate interface. The interface code library is placed between the EAS III application code and operating system. Similar to a "kernel", the library shielding those functions that may depend on the hardware and operating system from the core application code. The modular design allowed service-specific subsystems to be used for each branch of the armed services. There are specific subsystems for the Army, Air Force, and Navy.

5.2.2 DATA MANAGEMENT

Software tools were developed by NDC and deployed by EDS to design, document, and create the EAS III data dictionary.

5.2.3 CODING STANDARDS

The developers of EAS III use naming conventions and standards to control the development process. The developers use the NDC CASE tools to generate the code for all display screens and half of the printed reports.

Style guidelines were established for EAS III programming. All MUMPS code written conforms to the 1990 ANSI Standard. Block structured programming techniques and subroutines are used. Go To instructions are minimized and are only used as a last resort. Automated syntax checkers verify that MUMPS code conforms to EDS standards.

5.2.4 MAINTENANCE

All Incident Reports and Software Changes Requests (SCRs) from the field are reported to Health Affairs. Health Affairs then issues a software change request to the EDS development and maintenance staff. EDS assigns a number to each SCR. The SCR is then assigned to a EDS systems engineer. The systems engineer determines the MUMPS code affected by the SCR and checks the code out of the baseline code library of EAS III programs. Programs checked out of the library are placed into a

separate storage area for editing. When the changes to the code are complete, the MUMPS code is then moved to another storage area for testing.

5.2.5 TESTING AND VERIFICATION

Several phases of testing and verification are employed. Unit testing is initially performed by the systems engineer who updates the MUMPS code; this testing ensures that the routine works properly by itself. The code is then re-tested by a systems engineer who did not write any of the code. If the code is acceptable, it is moved to another storage area for integration and system testing to ensure that it works with the other routines. After the code passes system testing, it is returned to the baseline code library.

5.2.6 RELEASE AND DISTRIBUTION

EDS handles the release and distribution of EAS III. Prior to release, each new version is acceptance-tested by representatives from Health Affairs and the services. If the new version is acceptable, then EDS makes a tape of the new version and distributes it to the central facilities for each of the services. Further deployment from the central site to the end sites is handled by the services. EAS III provides the MTFs a communications function that will electronically apply software updates from the service agencies.

5.2.7 DOCUMENTATION

When a new version of EAS III is released, EDS produces a new set of program documentation, installation notes, and release notes to accompany it.

5.3 EAS III CURRENT USAGE

EAS III is currently operating at about 200 sites, including all military hospitals. The planned implementation of EAS III is for a total of 221 sites. The current versions of EAS III operate in a Unix environment on both AT&T 3B2 minicomputers and Everex 486-based PC microcomputers provided by the individual services. The Army and the Air Force run EAS III on AT&T 3B2 minicomputers under Unix, and the Navy runs EAS III on 486-based microcomputers under SCO/Xenix. No other hardware platforms are under consideration at this time.

Under contract to DoD, EDS employs 25 systems engineers to develop EAS III programs and 13 programmers to maintain existing EAS III code. Service branches provide about 70% of end user support, and EDS provides the rest.

5.4 EAS III PLANNED USAGE

The EAS III program is a recent enhancement that provides reporting at an MTF level to support local decision-making and planning. EAS III is an integral part of the MEPRS program. Local users will increasingly rely in EAS III in the future for its ability to provide improved decision support tools.

5.4.1 MUMPS ALTERNATIVES

The current plans for EAS III are to continue to use MUMPS as the development language. The EAS III Program Office is planning to evaluate conversion from MUMPS to Ada and SQL in the future.

5.4.2 MUMPS PORTABILITY

EDS has experienced some problems in migrating MUMPS code from the Micronetics MUMPS used for development to the Intersystems MUMPS used in the target platform. In particular, the implementation of the MUMPS string extract function (\$EXTRACT) differs slightly between these two versions. The \$EXTRACT function is used to extract individual or series of characters from a character string. This function is widely used in EAS III programs.

6.0 CHCS

The Composite Health Care System (CHCS) is a modular, integrated Hospital Information System that DoD is preparing to deploy at DoD medical facilities worldwide. CHCS is designed to provide worldwide support in key functional areas to all DoD Medical Treatment Facilities (MTFs). Health Affairs has contracted with Science Applications International Corporation (SAIC) to develop CHCS. SAIC adopted the VA's Decentralized Hospital Computer Program (DHCP) and modified it to meet DoD specifications.

6.1 CHCS BACKGROUND

The CHCS medical information system integrates automated data management to provide better and faster care of patients at DoD MTFs while containing costs and managing resources. CHCS supersedes manual and non-integrated AISs that are currently in use. CHCS gives doctors access to full medical records and integrates pharmacy, laboratory, radiology, nursing, appointment scheduling, patient administration, inpatient clinical services, outpatient clinical services, and other ancillary services (pharmacy, radiology, etc.).

CHCS was developed with input from all three services. CHCS subsumes the functionality of the Tri-Service Medical Information Systems (TRIMIS) pilot systems: Integrated Hospital Information Systems, and the Initial Operating Capability (IOC) systems for pharmacy, radiology, laboratory, patient appointments, patient scheduling, and patient administration. The IOC systems will be replaced by CHCS when it is deployed at DoD MTFs.

In 1988, DoD awarded a contract to SAIC to develop, test, deploy, and support CHCS. SAIC's proposal was based on the Department of Veterans Affairs (VA) Decentralized Hospital Computer Program (DHCP). DHCP automates almost all of the VA's information tracking and reporting functions, with integrated appointment scheduling, financial accounting, patient care tracking, inventory and resource tracking, clinical support, and management reporting modules that support almost every business area of the VA. Unlike DHCP, which is based on ancillary services, CHCS is centralized around order entry. The CHCS order entry emphasis tracks and tags orders throughout the system, allowing automated scheduling functions to be added.

6.1.1 ORDER ENTRY/RESULTS REPORTING

The Order Entry/Results Reporting system is the core of the CHCS system. Data is collected from the source as a by-product of the patient care process. The Order Entry/Results Reporting systems integrates will all CHCS functions, resulting in an integrated patient database.

6.1.2 PATIENT ADMINISTRATION

The Patient Administration system supports patient registration, admission, disposition, transfer and inpatient records functions. The Patient Administration module enables the health care worker to electronically access inpatient medical records, monitor patient status and test results, and review patient demographic or medical history data.

6.1.3 PATIENT APPOINTMENT AND SCHEDULING

The Patient Appointment and Scheduling system supports a centralized appointment service with a decentralized appointment scheduling option. The system tracks all patient encounters within the MTF and monitors and maintains clinic schedules.

6.1.4 LABORATORY

The Laboratory system supports laboratory services with patient identification, patient registration, order processing, specimen processing, results management, blood bank control, laboratory management, tumor registration, and inventory management.

6.1.5 RADIOLOGY

The Radiology system supports radiology with order processing, patient inquiry processing, film inventory management, patient eligibility checking, and administrative functions.

6.1.6 PHARMACY

The Pharmacy system supports pharmacy operations with on-line drug monographs, drug interaction, allergy, drug age range/dosage verification, order processing and administrative functions.

6.1.7 NURSING

The Nursing system supports a variety of nursing functions, including documenting and performing verifications, entering and inquiring on orders, entering patient diagnosis data, generating patient care plans, and documenting patient progress towards goals. Administrative functions in the areas of staffing and workload management are also supported.

6.1.8 CLINICAL DIETETICS

The Clinical Dietetics system supports the MTF food service to identify patient nutritional needs, establish patient dietary plans, process patient diet orders, maintain patient nutritional data, compute Dietetics workload data. The Clinical Dietetics system integrates with the Tri-Service Food Service Program (TRIFOOD).

6.1.9 QUALITY ASSURANCE

CHCS will support MTF quality assurance operations, superseding AQCESS. Functions will include quality assurance incident monitoring, incident disposition and reporting.

6.1.10 MOBILIZATION AND MASS CASUALTY OPERATIONS

CHCSs ability to operate as a worldwide integrated medical information system supports the DoDs special requirement for mobilization and mass casualty operations at MTFs. The CHCS program will support coordinated mobilization and mass casualty operations through communication capabilities being built into CHCS.

6.2 CHCS PROGRAM

The role of CHCS is to improve the availability of clinical and administrative information within an MTF. CHCS assists providers and administrators by supporting a wide range of functions. CHCS provides benefits for health care administrators, health care providers, ancillary staff, patients, MTFs, and DoD health program managers. Health care providers are provided with extensive and immediate access to

current and historical patient information and clinical reference material. Data will be available to the health care provider in the formats useful for making clinical judgments. CHCS reduces paperwork, automates scheduling, transmits messages, automatically marks test results, and reduces unnecessary testing due to better patient data availability.

The CHCS deployment strategy, referred to as "Modular CHCS", specifies the CHCS modules that are to be initially deployed at all MTFs. The Modular CHCS programs support the core functions performed at all MTF's and clinics and consist of the following modules:

- Order Entry/Results Reporting
- Patient Administration
- Patient Appointment and Scheduling
- Laboratory
- Pharmacy
- Radiology

After initial deployment, additional functionality can be added to the CHCS system. Additional CHCS support for MTF functions is implemented or planned to be implemented in the following areas:

- Nursing
- Clinical Dietetics
- Quality Assurance
- Mobilization and Mass Casualty Operations

Starting in 1995 these modules are scheduled to be deployed at the 14 largest MTF's. The modular structure of CHCS allows MTFs to pick and choose those modules that will support their specific functional needs.

6.3 CHCS DEVELOPMENT

In developing CHCS, SAIC has established software development standards and policies to better manage the development of MUMPS software. CHCS is based on the VA's DHCP system which was written in the MUMPS programming language. Because MUMPS is interpreted instead of compiled, SAIC has found that rapid application development is possible, with rapid prototyping so that the programmers can modify the program much more quickly to meet customer needs.

Software development and deployment transpires concurrently in the CHCS program. CHCS software is deployed incrementally at Alpha and Beta sites. Software upgrades are deployed at Alpha sites as they are developed.

6.3.1 ARCHITECTURE

As in DHCP, CHCS uses a modular architecture with application modules that interact with a "Kernel" instead of the underlying hardware and operating system. The Kernel serves to insulate the application code from the underlying hardware and operating system. Unlike DHCP, which is decentralized and oriented toward ancillary services, CHCS is centralized around order entry. All application modules are routed through the order entry for tracking and tagging of orders and results. This gives CHCS the potential to perform automated scheduling, appointing, and routing.

6.3.2 DATA MANAGEMENT

CHCS uses a FileMan compatible database for data access and storage. A Current disadvantage with MUMPS database structure is that the VMS operating system recognizes a FileMan database as one large file. VMS operating system utilities to defragment files will not work on the FileMan data base. Digital Equipment is currently working on a tool with the ability to defragment MUMPS databases.

6.3.3 CODING STANDARDS AND DEVELOPMENT

Implementation-specific extensions (those not found in the ANSI specifications) are not allowed in CHCS, although some MDC-approved Type A extensions are used. SAIC uses MUMPS software tools developed by the VA for screen design, report generation, code checking, messaging, program job control and messaging. A government developed tool is being used to validate and verify software released for testing and installation.

6.3.4 MAINTENANCE

SAIC stated that maintenance of MUMPS code, in comparison to more structured languages such as Ada, can be difficult without a standard set of programming tools, programming standards, and conventions to develop modular structure code for future reuse. However, debugging is easier because MUMPS is an interpreted language; the current variables and line that was in error are immediately available when an error occurs.

6.3.5 TESTING AND VERIFICATION

DoD has established a developmental testing and evaluation organization (DT&E) which is responsible for software testing and validation prior to implementation at an MTF Alpha site. DT&E personnel are involved in the entire software development process; reviewing initial requirements, validating software design and design documentation, and validating SAIC test plans and testing methods. After successful completion of software testing by SAIC, DT&E personnel perform pre-installation acceptance testing in the form of unit and integration testing. Alpha site implementation is scheduled upon acceptance of the software by DT&E.

Alpha site testing of CHCS programs ensures that the program operates as originally intended and is suitable for use under the Operational Testing and Evaluation program (OT&E) at 14 Beta sites. The purpose of OT&E is to measure the effectiveness and suitability of CHCS prior to any decision to deploy. The Beta site MTFs comprise approximately 20% of the total MTF workload. Extensive testing under operational conditions is performed at the Beta sites by the Office of Health Systems Evaluation. Test results are forwarded to the Operational Test and Evaluation Review Group (OTERG) for review and approval. OTERG is responsible for certifying that the test results support worldwide deployment.

6.3.6 RELEASE AND DISTRIBUTION

MTF's must be nominated for installation of CHCS by Military Services. The Military Department of the MTFs service branch organizes a deployment team to work closely with the MTF and CHCS Program Office in planning the implementation of the system. After implementation it is the responsibility of SAIC to certify the system for government use. A government pre-installation acceptance test (GPIAT) commences consisting of system performance and functional tests performed by the government. During this period the government certifies that the system meets hardware and software performance requirements according to contract specifications.

6.4 CHCS CURRENT USAGE

CHCS has been deployed at two Alpha site hospitals and twelve Beta site hospitals which completed (OT&E) in May 1992. CHCS is currently deployed on clusters of Digital Equipment Corporation (DEC) VAXes running Digital Standard MUMPS (DSM) V6.0. VAX clusters can include VAX 4000 series, VAX 6000 series, VAX 8550, and microVAX 3000 series. In addition, MS-DOS microcomputers running DataTree MUMPS V4.4 and Unix systems running Intersystems M/SQL V4.5 are used for CHCS.

SAIC employs 100 programmers working on CHCS development; in addition, Health Affairs estimates an equal number of people support CHCS development.

6.5 CHCS PLANNED USAGE

The DoD MAISRC approved CHCS at the Milestone IIIA review 19 May 1992. The Milestone IIIA review approved a DoD-wide deployment of the CHCS subsystems which replace obsolete IOC systems. DoD is preparing to deploy CHCS to all military hospitals worldwide pending Congressional approval. CHCS deployment will occur in phases over a 7 year period.

The modular structure of CHCS has allowed the DoD to develop a deployment plan ("Modular CHCS") for its 155 hospitals and 36 clinics that accelerates the implementation of basic CHCS functions to the larger MTF's. This eliminates the need for sequential site by site deployment of the full CHCS program. The current plan provides for the basic functions of CHCS to be deployed at all MTFs with fifty beds or more by 1995. CHCS clinical service functions will be added at the 14 largest MTF's starting in 1995. DoD intends for CHCS eventually to support all military MTF clinical and administrative functions worldwide.

6.5.1 MUMPS ALTERNATIVES

SAIC believes that "MUMPS is ideally suited to string manipulation" because it has very powerful string parsing and string handling functions; also, all MUMPS data are character strings. MUMPS integrated powerful string manipulation and B-tree data storage support powerful text handling and rapid access to data. SAIC finds that MUMPS is a good choice for the heavy text processing that dominates Hospital Information Systems.

6.5.2 MUMPS ENHANCEMENTS

The CHCS office has identified several areas of improvement for the MUMPS environment across several areas:

- Software development tools
- Open Systems Environment support
- Program maintenance and utilities

6.5.2.1 Software Development Tools

Health Affairs and SAIC would like to have more integrated CASE tools which support forward and reverse engineering, to assist with development of MUMPS programs. Software tools for application design and user interface that supports a window-based Graphical User Interface (GUI) is strongly desired. SAIC would like to see object-oriented programming concepts incorporated into MUMPS.

6.5.2.2 Open Systems Environment support

Both SAIC and Health Affairs want improved interfaces to Open Systems in the form of bindings to SQL, X-Windows and other standards. The extensions proposed by the MDC for the 1993 ANSI Language Standard will greatly improve the ability of MUMPS to support industry standards.

6.5.2.3 Maintenance programs and Utilities

Health Affairs also sees a need for tools to maintain MUMPS data bases so that the data base can be defragmented and tuned for better performance. Enhanced error trapping and error recovery capabilities would be desirable. More automatic routine management and testing tools would also be useful. SAIC believes that incorporation of PC MUMPS Programming Support Environments for development would be desirable.

7.0 SUMMARY

This report documents the Department of Defense (DoD) usage of the MUMPS programming language. The information in this report was gathered by canvassing staff within DoD and under contract to DoD, to determine MUMPS usage. The MUMPS User Group (MUG) was consulted to gather information about the MUMPS programming language.

DoD and its contractors have found MUMPS to be an effective language for developing their HISs. DoD uses MUMPS as the language for HISs that automate aspects of its hospitals such as workload management and quality of care reporting. The DoD is preparing to field a comprehensive MUMPS-based HIS that will integrate and automate all data management for a hospital. DoD is evaluating conversion to Ada and SQL for systems still under development in order to conform with Open Systems Environment guidelines and policies. DoD and its contractors have noted a number of areas in which MUMPS could be improved.

7.1 DATA GATHERING METHODS

The information presented was gathered by canvassing experts within the DoD and its contractors on both the current and planned usage of the MUMPS programming language. Telephone interviews with DoD staff members, surveys sent to DoD and contractor staff, and personal interviews were all used to determine information about DoD's usage of MUMPS. Additional information was gathered from the MUG, the MDC, reference materials on MUMPS based AISs, and the MDC's MUMPS Language Standard.

7.2 MUMPS BACKGROUND

The Massachusetts General Hospital Utility Multi-Programming System, commonly referred to as "MUMPS", was initially developed by the Laboratory of Computer Science of Massachusetts General Hospital in 1967. MUMPS is both an interactive programming language and a general purpose database management system that has been widely used in the development of medical information systems.

The MDC publishes both a MUMPS Language Standard and a MUMPS Portability Standard. The MUMPS Language Standard has been approved as a standard programming language by the FIPS Standard, the ANSI, and by the International Standardization Organization (ISO). The MUMPS Portability Standard allows

conforming application code to be truly portable among different MUMPS implementations, operating systems, and hardware platforms.

Although MUMPS was originally developed for minicomputers, versions of MUMPS are currently available for mainframes, minicomputers, workstations, and personal computers. MUMPS is best known for its ability to handle unstructured textual information, its data sharing capabilities, and the interpretive nature of the language.

7.3 DOD MUMPS USAGE

DoD contractors use the MUMPS programming language in developing and maintaining medical and health information systems. DoD has deployed MUMPS-based systems at all of its hospitals, using them to automate various functions at each hospital. DoD plans to deploy a new comprehensive MUMPS-based system to replace older initial automation systems at its hospitals. DoD and its contractors have noted areas in which MUMPS could be improved to better support DoD requirements.

7.3.1 DOD MUMPS DEVELOPMENT

DoD contractors developing AQCESS, EAS III, and CHCS all follow standard industry practice for software development. All of the DoD contractors controlled both the process by which the software was developed and also the format and style of the software. It is notable that all of the developers stated that it was important to properly control the development of MUMPS code. All of the developers established and adhered to conventions and standards because MUMPS is an unstructured language which allows commands to be abbreviated to a single letter, commands to be combined on a single line, and data to be executed as code; otherwise, the resultant code may be difficult to maintain. All of the developers used CASE tools and MUMPS Programming Support Environments to manage code development, maintenance, and release. Naming conventions were required by all developers because variables are shared by all users and all routines. Because MUMPS is not compiled, all developers used syntax checkers to verify that their code conformed to ANSI Standards. Code generators were also used to create screens and reports.

7.3.2 DOD CURRENT USAGE

DoD has a major investment in MUMPS, using it for AISs that automate some functions in all of its hospitals. DoD depends on MUMPS for automating quality assurance all of its hospitals through the Automated Quality of Care Evaluation Support System (AQCESS) and other pilot projects. DoD has deployed AQCESS at

about 160 sites, which includes all military hospitals and the largest military clinics. DoD currently depends on the Expense Assignment System, version III (EAS III), a MUMPS-based tool system that provides consistent uniform reporting of expenses and manpower, as its only workload management tool. EAS III has been deployed at about 200 sites, and will eventually be deployed at 221 sites.

DoD is preparing to deploy the Composite Health Care System (CHCS), a MUMPS-based Hospital Information System (HIS), that will replace all of the MUMPS pilot systems. CHCS was deployed to 2 Alpha sites and 11 Beta sites for testing. The DoD Major Automated Information Systems Review Committee (MAISRC) has approved CHCS deployment to all military hospitals, pending Congressional approval.

7.3.3 DOD MUMPS PLANNED USAGE

The DoD has published a Technical Reference Model (TRM) which outlines DoD's policy of migration towards an Open Systems Environment (OSE). The DoD TRM is based on the National Institute of Standards and Technology (NIST) Application Portability Profile (APP). The TRM and APP specify that standards that should be used for application software within DoD and the U.S. government, respectively. The MUMPS programming language combines programming language services with database management functions; the DoD TRM specifies Ada as the approved programming language and SQL as the database manager.

Both EAS III and CHCS, which have development ongoing, are under consideration by Health Affairs for conversion to Ada and SQL. AQCESS, which has already been deployed and is being maintained has been considered for conversion to Ada and SQL. Health Affairs has determined, however, that conversion would not be cost effective since the system is to be replaced by CHCS.

7.3.4 DOD MUMPS ENHANCEMENTS

DoD and its contractors identified security, standards, interface, and performance-related changes that they would like to see made to the MUMPS language. A more secure language would separate the source code from the executable code so that a malicious user could not have access to the source code. Improved support of standards, such as SQL and X-Windows, would promote integration of MUMPS in an Open Systems Environment. Support for export of MUMPS data to other systems, for data interchange and integration, would result in improved data communications. Additional performance-tuning tools are desired by DoD.

APPENDIX A.1: ABBREVIATIONS

| <u>Acronym</u> | <u>Meaning</u> |
|----------------|---|
| AIS | Automated Information System |
| ANSI | American National Standards Institute |
| AQCESS | Automated Quality of Care Evaluation Support System |
| CHCS | Composite Health Care System |
| DBMS | Data Base Management System |
| DEERS | Defense Enrollment Eligibility Reporting System |
| DHCP | Decentralized Hospital Computer Program |
| DoD | Department of Defense |
| DRG | Diagnosis Related Group |
| DSS | Decision Support System |
| EAS III | Expense Assignment System, version III |
| EDS | Electronic Data Systems |
| FIPS | Federal Information Processing Standard |
| GAO | Government Accounting Office |
| GKS | Graphical Kernel Services |
| GUI | Graphical User Interface |
| HA | Health Affairs |
| HIS | Hospital Information System |
| IEC | International Electrotechnical Committee |

| | |
|---------|--|
| IHS | Indian Health Service |
| IRDS | Information Resource Dictionary System |
| ISO | International Standards Organization |
| MAISRC | Major Automated Information Systems Review Council |
| MDC | MUMPS Development Committee |
| MUG | MUMPS Users Group |
| MUMPS | <u>M</u> assachusetts General Hospital <u>U</u> tility <u>M</u> ulti- <u>P</u> rogramming <u>S</u> ystem |
| NDS | National Data Corporation |
| OMI | Open MUMPS Interconnect |
| OT&E | Operational Test and Evaluation |
| RDBMS | Relational Database Management System |
| SAIC | Science Applications International Corporation |
| SIDDOMS | Systems Integration, Design, Development, Operations, and Maintenance |
| SQL | Structured Query Language |
| TCP/IP | Transmission Control Protocol/Internet Protocol |
| VA | Department of Veterans Affairs; was Veteran's Administration |
| VAX | Virtual Addressing Architecture |

APPENDIX A.2: REFERENCES

A.2.1 MUMPS LANGUAGE STANDARDS AND SPECIFICATIONS

The Complete MUMPS: An Introduction and Reference Manual for the MUMPS Programming Language. John Lewkowicz, Prentice Hall. 1989.

ABCs of MUMPS. Dr. Richard F. Walters, Digital Press. 1989.

ANSI/MDC X11.1 - 1984. Mumps Development Committee.

A.2.2 MUMPS BASED AISs

Composite Health Care System (CHCS) System Decision Paper. Defense Medical Systems Support Center. May 1991.

The Functional Description (FD) For the Expense Assignment System Version III (EAS III). Electronic Data Systems, 1989.

Medical Expense and Performance Reporting System (MEPRS) and Expense Assignment System, Version III (EAS III), Data Base Specification, Version 5.0. Defense Medical Systems Support Center. September 1992.

Program Fact Book. Office of the Deputy Assistant Secretary of Defense (Health Management Systems). January 1990.

APPENDIX A.3: MUMPS SURVEY

MUMPS Survey

1. What is your name, title, telephone number, and facsimile number?
2. What are the current host platforms for your MUMPS based AISSs?
3. What version and release of MUMPS is currently being used to maintain and develop your AISSs (e.g. VAX DSM v5.0)?
4. What implementation specific extensions of MUMPS (not defined in the ANSI specifications) are widely used in your AISSs?
5. How many programmers (in-house and contract) develop and maintain these systems? What percentage of development and maintenance work is performed by your staff and what percentage by contractors?
6. What functional upgrades or other plans have been made with respect to your MUMPS AISSs? What other programming environments and languages are you considering for future maintenance and development?

MUMPS Survey

7. What other programming languages interface with your MUMPS systems and could be used for future development?
8. What advantages does MUMPS provide over other languages?
9. What disadvantages does MUMPS have in comparison to other languages?
10. What other programming environments and languages do you have direct experience with or knowledge of?
11. Based on your experience with the environments and languages in 7, how does MUMPS compare to each of them with respect to the following, and why?
 - a. productivity
 - b. maintainability
 - c. usability
 - d. portability
 - e. functionality

MUMPS Survey

12. Based on your experience with software development in MUMPS, what changes would you like to see in the MUMPS programming language to help you to meet future user requirements?
13. Please provide additional comments you may have regarding MUMPS not directly addressed in previous questions.

APPENDIX A.4: MUMPS SURVEY RESPONSES

APPENDIX A.5: MUMPS INTERVIEW QUESTIONNAIRE

Interview Questionnaire

SUMMARY

The Office of Technical Integration has requested a study of the MUMPS programming language and DoD automated information systems (AISs) written in MUMPS that support the medical business area. Specifically, this study will gather background data on the MUMPS programming language and how it is used. In support of this objective, the following questionnaire provides a summary of the variety of information needed by the study. A personal interview will be conducted to discuss these items in more detail.

1. What are the current host platforms for your MUMPS based AISs?
2. What version and release of MUMPS is currently being used to maintain and develop your AISs?
3. What implementation specific extensions of MUMPS (not defined in the ANSI specifications) are widely used in your AISs?
4. What functional upgrades or other plans have been made with respect to your MUMPS AISs? What other programming environments and languages are you considering for future maintenance and development?
5. What advantages and disadvantages in meeting user functional requirements does MUMPS provide in comparison with other languages?
6. How many programmers (in-house and contract) develop and maintain these systems? What percentage of development and maintenance work is performed by your staff and what percentage by contractors?
7. Based on your experience with other programming environments and languages, how does MUMPS compare to them with respect to: productivity, maintainability, usability, portability and functionality?
8. Based on your experience with software development in MUMPS, what changes would you propose in future specifications of the MUMPS programming language to improve your ability to meet future user and market requirements?

APPENDIX A.6: MUMPS INTERVIEW

Date: 20 October 1992

Location:

Falls Church, Virginia

Attendees: MAJ Tony Barnett -- CHCS PO
Nancy Hoffer -- OTI
Mike Sinisgalli -- EDS
Gerry Bragg -- VRI
Roger Christensen -- VRI
John Hwang -- VRI

Transcript:

Hardware Platform

Health Affairs:

- Vax 6000's (stand-alone and clustered)
- Vax 8550's
- MicroVax 3000
- Future: Micro and RISC based

SAIC:

- IBM, Data General, Pyramid

MUMPS versions

- DSM 5.0 (will upgrade to 6.0a)
- Datatree 4.3, 4.4 (PC)

MUMPS portability

- Use "Z" calls which may be implementation specific
- Any calls for system level utilities, devices obviously won't port.

SAIC indicated they didn't use any extensions in their programs.

Other Languages considered

- Ada, 4GL, C++ & other OOP's

CASE tools

SAIC uses many VA tools (syntax checker, TaskMan, MailMan). Also designed screen, report generators (or uses VA tools?). Many commercial tools are

available for optimization, program documentation, GUI (Hyper-M, Cyber-M), SQL queries (M-SQL)

M-SQL is FileMan compatible.

Future MUMPS tools (desired)

- Development tools - windows (GUI); SQL, X-Windows bindings, I-Case
- True mumps compiler
- Data base performance enhancer (defragmenting,...) VMS can't recognize FileMan structure. Need VMS level tool that will defragment/re-organize MUMPS data base.

Advantages of MUMPS

- String manipulation features
- Hospital IS's are primarily DBMS's.
- As a DBMS MUMPS is:
 - Good at storing large amounts of data (variable length records make efficient use of storage space)
 - Competitive performance
 - Has many features to optimize access to large databases.
- Interpretive nature of language supports rapid prototyping, maintenance, testing and debugging very well.
 - Mean time to repair bug can be 15 minutes.
- Learning curve for language is very short.

Disadvantages of MUMPS

- Not secure. Code can be changed or other commands inserted during execution. (Interpretive 'drawback'). Additional security is required to block access to program code.
- No good statistical functions.
- Cryptic nature of language
 - Use of indirection
 - Multiple statements per line.

CHCS security

Security established on user level through VMS internally. MUMPS built security programs assist in securing on a record and element level (since VMS sees FileMan data as one large file. Can't recognize records and elements). Uses MUMPS access control features.

MUMPS and AST definition

Productivity

Advantage in startup time. APL is only higher productivity language.

Maintainability

MUMPS can be a problem. There are no enforced conventions, structured programming enforcements, programmer reference documentation enforcements. If these are established and enforced manually, MUMPS is just as maintainable.

Usability

Rapid prototyping capability enhances usability. Extremely usable.

Functionality

Can make a MUMPS program do basically anything you want. Depends on what language is supposed to do. Very functional as an application development tool and DBMS. Powerful aggregates of commands can be constructed.

**ADVANCED SOFTWARE TECHNOLOGY PLANS
FOR MUMPS FIPS PUB**

TABLE OF CONTENTS

| | PAGE |
|---|------|
| 1.0 INTRODUCTION | 1-1 |
| 1.1 Study Background | 1-1 |
| 1.2 MUMPS Background | 1-2 |
| 1.3 FIPS Background | 1-3 |
| 1.4 DoD Standards Policy | 1-3 |
| 1.5 Document Organization | 1-3 |
| 2.0 AST CONTEXT | 2-1 |
| 2.1 AST Applied to Software Engineering | 2-1 |
| 2.2 Other Criteria | 2-4 |
| 3.0 MUMPS BACKGROUND | 3-1 |
| 3.1 Distinctive Characteristics of MUMPS | 3-1 |
| 3.2 MUMPS Standardization | 3-9 |
| 4.0 CURRENT FIPS PUB | 4-1 |
| 4.1 1984 ANSI Standard | 4-1 |
| 4.2 Differences Between FIPS PUB and 1984 ANSI Standard | 4-3 |
| 5.0 CURRENT MUMPS STANDARD AND PLANNED FIPS PUB | 5-1 |
| 5.1 Block Structuring | 5-1 |
| 5.2 Variable Scoping | 5-2 |
| 5.3 Formatting Functions | 5-5 |
| 5.4 Search Functions | 5-6 |
| 5.5 MUMPS FIPS PUB Status | 5-8 |
| 6.0 PROPOSED 1993 ANS | 6-1 |
| 6.1 Proposed Changes to MDC MUMPS Language Standard | 6-1 |
| 6.2 Proposed Changes to Portability Standard | 6-9 |
| 6.3 Status of MDC MUMPS Language Standard | 6-10 |
| 7.0 SUMMARY | 7-1 |
| 7.1 Applicability of AST Criteria to MUMPS | 7-1 |
| 7.2 Conclusions | 7-6 |

TABLE OF CONTENTS

| | PAGE |
|-------------------------|------|
| A.0 APPENDICES | |
| A.1 Abbreviations | A1-1 |
| A.2 References | A2-1 |

LIST OF EXHIBITS

| <u>Number</u> | <u>Title</u> | <u>Page</u> |
|---------------|---|-------------|
| 3-1 | Distinctive Characteristics of MUMPS | 3-2 |
| 3-2 | MUMPS Symbolic Operators | 3-3 |
| 3-3 | One-Character Commands in MUMPS | 3-4 |
| 3-4 | The Tree Structure of MUMPS Global Arrays | 3-8 |
| 3-5 | Timeline of MUMPS Standardization | 3-10 |
| 3-6 | The MUMPS Standards Process | 3-12 |
| 4-1 | Example of String Subscripts | 4-2 |
| 4-2 | Example of \$Order Function | 4-2 |
| 4-3 | Example of Subscript Indirection | 4-3 |
| 5-1 | Example of Block Structuring | 5-2 |
| 5-2 | Example of New Command Syntax | 5-3 |
| 5-3 | Example of Parameter Passing Syntax | 5-4 |
| 5-4 | Example of Extrinsic Function Call | 5-5 |
| 5-5 | Example of Translate Function | 5-6 |
| 5-6 | Example of \$Fnumber Function | 5-6 |
| 5-7 | Example of \$Get Function | 5-7 |
| 6-1 | Calling External Routines from MUMPS | 6-2 |

1.0 INTRODUCTION

The Office of Technical Integration (OTI) has requested a study of the Massachusetts General Hospital Utility Multi-Programming System (MUMPS). This report focuses on one aspect of the study: plans to add Advanced Software Technology (AST) capabilities to the MUMPS Federal Information Processing Standard (FIPS). The MUMPS FIPS Publication (FIPS PUB 125) guides the use of the MUMPS programming language within the government, specifying minimum requirements that vendors of MUMPS interpreters must meet for Federal use of their products.

The Department of Defense (DoD) requires the use of Ada or other technologies that meet the definition of an AST for development and major enhancement of information systems. This report documents the development of both the MUMPS programming language and the MUMPS FIPS PUB, as well as the changes planned for each standard within the context of AST. The information in this report was gathered from the MUMPS User's Group (MUG), the MUMPS Development Committee (MDC), and the Department of Commerce National Institute of Standards and Technology (NIST).

Both the FIPS PUB 125 and the MUMPS programming language are in a state of transition. The FIPS PUB is being revised to reflect the current 1990 MUMPS Language Standard, instead of the 1984 Standard. The MUMPS Language Standard itself is being enhanced to support a more open architecture; interfaces have been proposed to support Open Systems, transaction processing, and data communications in a 1993 revision of the Standard. The revisions of the FIPS PUB and the Language Standard are both in the final comment stages; they will be finalized in 1993 or 1994.

1.1 STUDY BACKGROUND

DoD Directive (DoDD) 3405.1, "Computer Programming Language Policy," states that DoD policy is to prefer, based on life-cycle costs and impact, use of:

- Off-the-shelf application packages and advanced software technology (AST)
- Ada-based software and tools
- Approved standard Higher Order Languages (HOLs)

The Assistant Secretary of Defense reinforced this policy in a memorandum dated 17 April 1992. This memorandum, titled "Delegations of Authority and Clarifying Guidance on Waivers from the Use of the Ada Programming Language," required all new development or modification of DoD software to comply with DoDD 3405.1.

The DoD requires a waiver for development of software, or modifications to more than one-third of the existing code, that will not use the Ada programming language. This waiver must be supported by documentation demonstrating that the benefits of the AST definition are met. The definition of AST given in DoDD 3405.1, was clarified in the 17 April 1992 memorandum to be:

Software tools, life-cycle support environments (including programming support environments), non-procedural languages, modern database management systems, and other technologies that provide significant improvements in productivity, usability, maintainability, portability, etc., over those capabilities commonly in use.

This study examines the changes planned for the MUMPS FIPS PUB and programming language with respect to the AST definition. MUG and the MDC were consulted about the development and planned changes in the MUMPS programming language. NIST was contacted for information about development of and changes to FIPS PUB 125.

1.2 MUMPS BACKGROUND

MUMPS was initially developed by the Laboratory of Computer Science at Massachusetts General Hospital in 1967. MUMPS is both an interactive programming language and a general purpose database management system that has been widely used in the development of medical information systems.

The MDC publishes both a MUMPS Language Standard and a MUMPS Portability Standard. The MUMPS Language Standard has been approved as an American National Standard (ANSI/MDC X11.1-1990), a FIPS Publication (FIPS PUB 125), and an International Standards Organization (ISO) Standard (ISO/IEC 11756). The MUMPS Portability Standard allows conforming application code to be truly portable among different MUMPS implementations, operating systems, and hardware platforms.

Although MUMPS was originally developed for minicomputers, MUMPS versions are currently available for mainframes, minicomputers, workstations, and personal computers. MUMPS is best known for its ability to handle unstructured textual information, its data sharing capabilities, and the interpretive nature of the language.

1.3 FIPS BACKGROUND

NIST is responsible for improving the use and management of computer systems used in the Federal Government. Through its Computer Systems Laboratory, NIST

provides leadership and technical guidance to government in the development of computer systems standards and guidelines. FIPS PUBS are guidelines and standards for technical subjects issued by NIST after approval by the Secretary of Commerce.

1.4 DOD STANDARDS POLICY

Within DoD, the usage of DoD Standards and Military Standards (MIL-STD) is mandatory if either standard exists. A FIPS PUB would be used if neither DoD nor MIL-STD exists. Otherwise, the usage of an American National Standard (ANS) is recommended if it does not conflict with an existing FIPS PUB or DoD Standard or MIL-STD.

1.5 DOCUMENT ORGANIZATION

The remainder of this document is organized as follows. Chapter 2.0 defines the terms used in the definition of AST more completely. These definitions will form the basis for the analysis of MUMPS in the later chapters. Chapter 3.0 provides background on the features of MUMPS as a language and as a data manager; it also describes the initial efforts to standardize MUMPS. This background will be the basis for explaining the current standards and future plans for MUMPS.

Chapters 4.0 through 6.0 describe the current and proposed standards for MUMPS and how the features in each standard relate to the criteria for AST. Chapter 4.0 discusses the current MUMPS FIPS PUB, which is based on the 1984 ANS. Chapter 5.0 describes the enhancements introduced in the 1990 ANS that will be reflected in the next revision of the FIPS PUB. Chapter 6.0 then describes changes that are being planned for the next revision of the ANS, targeted for 1993. Chapter 7.0 summarizes the previous chapters and analyzes the current and proposed versions of MUMPS from the viewpoint of the criteria of the AST.

Appendix A.1 lists the abbreviations and acronyms used in this report. Appendix A.2 lists the references used in preparing this report.

2.0 AST CONTEXT

This study provides information on the relationship between AST and the changes planned to the MUMPS FIPS PUB and Language Standard. Department of Defense policy states that all non-Ada development and maintenance must be done under a waiver that documents how the alternative chosen achieves the benefits of AST. Usage of the MUMPS programming language may require such a waiver. Determining whether the MUMPS programming language is moving toward or away from the AST definition will be useful in preparing and evaluating such waivers.

As noted in chapter 1.0, DoD Directive 3405.1 defines AST as follows:

Software tools, life-cycle support environments (including programming support environments), non-procedural languages, modern data base management systems, and other technologies that provide significant improvements in productivity, usability, maintainability, portability, etc., over those capabilities commonly in use.

It is necessary to define the terms used in this definition to remove ambiguity about their meanings. Because DoD Directive 3405.1 does not define the terms used in the definition of AST, this chapter defines the terms within the context of software engineering in general, and with respect to programming languages in particular. The changes to the MUMPS programming language can then be evaluated with respect to the AST definition.

2.1 AST APPLIED TO SOFTWARE ENGINEERING

The terms used in the definition of AST require technical definitions which are more precise than those from common usage. The Institute of Electrical and Electronics Engineers (IEEE) defines many of these terms in its *IEEE Standard Glossary of Software Engineering Terminology* (IEEE Std 610.12-1990). This glossary defines terms in the AST definition as they are currently used within the field of Software Engineering. Those terms that were not defined in the glossary are defined below based on common practice and usage within industry.

2.1.1 IEEE DEFINITIONS

The *IEEE Standard Glossary of Software Engineering Terminology* was approved by ANSI in 1991. The definitions of the following terms used in the definition of AST are taken from the glossary:

- Software Tool
- Programming Support Environment
- Non-procedural Language
- Usability
- Maintainability
- Portability

2.1.1.1 Software Tool

A software tool is a computer program used in the development, testing, analysis, or maintenance of a program or its documentation. Examples include a comparator, cross-reference generator, decompiler, driver, editor, flowcharter, monitor, test case generator, or timing analyzer.

2.1.1.2 Programming Support Environment

A programming support environment (PSE) is an integrated collection of software tools accessed via a single command language to provide programming support capabilities throughout the software life cycle. The environment typically includes tools for specifying, designing, editing, compiling, loading, testing, configuration management, and project management.

2.1.1.3 Non-Procedural Language

A non-procedural language is one in which the user states what is to be achieved without having to state specific instructions that the computer must execute in a given sequence.

2.1.1.4 Usability

Usability is the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component.

2.1.1.5 Maintainability

Maintainability is the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment.

2.1.1.6 Portability

Portability is the ease with which a system or component can be transferred from one hardware or software environment to another.

2.1.2 NON-IEEE DEFINITIONS

Some of the terms used in the definition of AST were not defined in the IEEE glossary. Based on their common use within industry, the following terms used in the definition of AST are defined below:

- Life-cycle Support Environment
- Modern Database Management System
- Productivity

2.1.2.1 Life-Cycle Support Environment

A life-cycle support environment is an integrated set of software tools and utilities that are designed to support a software product from the time it is conceived until it is no longer used. A life-cycle support environment might automate functional and data requirements definition, functional and detailed design, development, testing, release, and maintenance.

2.1.2.2 Modern Data Base Management System

A modern data base management system (DBMS) supports the current state of the art, sharing data among users. The data is normalized so that the system contains no redundant data. This may be done through a hierarchical tree, defined links, or related tables. A modern DBMS supports transaction processing to ensure data consistency.

2.1.2.3 Productivity

Productivity is the ease with which systems, programs, and their component functions and procedures can be developed and written using the language. For example, a simple, commonly-used measure of productivity is the number of lines of code written per person in a given amount of time.

2.2 OTHER CRITERIA

In addition to the criteria directly noted within the AST definition, there are additional criteria which may apply to evaluations of MUMPS as AST. These criteria appear to be covered by the term "etc." in the AST definition. Three such criteria are defined in the IEEE glossary:

- Flexibility
- Efficiency
- Extendability

NIST's Application Portability Profile, the document that describes the government's Open Systems Environment, defines three other criteria:

- Completeness
- Maturity
- Stability

2.2.1 FLEXIBILITY

Flexibility is the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed.

2.2.2 EFFICIENCY

Efficiency is the degree to which a system or component performs its designated functions with minimum consumption of resources. The IEEE glossary defines two subcategories: execution efficiency, which is concerned with the minimal consumption of time, and storage efficiency, which is concerned with the minimal consumption of available storage.

2.2.3 EXTENDABILITY

Extendability is the ease with which a system or component can be modified to increase its storage or functional capacity.

2.2.4 COMPLETENESS

Completeness is the degree to which a standard defines and covers key features necessary in supporting a specific functional area or service.

2.2.5 MATURITY

Maturity is the degree to which a specification and its underlying technologies and concepts are well-understood. For example, the specification may be defined by a formal mathematical model or the technology may have been used for many years.

2.2.6 STABILITY

Stability is the likelihood that the specification will not have many or significant changes within the next two years.

3.0 MUMPS BACKGROUND

MUMPS was initially developed by the Laboratory of Computer Science of Massachusetts General Hospital in 1967. The objective of this effort was to develop an interactive, multi-user, minicomputer-based system that would support a hospital's special need to store, share, and manipulate textual data. Many of the characteristics of MUMPS were designed to support and manage the varied and sparse data typical of the medical environment. Through the years, MUMPS has evolved into a general purpose hierarchical database management system that can support a wide variety of applications, such as scheduling, finance, and inventory.

3.1 DISTINCTIVE CHARACTERISTICS OF MUMPS

MUMPS has several characteristics that differentiate it from other programming languages and data base managers. Exhibit 3-1 contains a description of the distinctive characteristics of MUMPS. The following sections discuss implications of each characteristic.

3.1.1 MUMPS AS A PROGRAMMING LANGUAGE

Many of the programming characteristics of MUMPS are distinct from those found in other languages. MUMPS did not evolve out of the mainframe environment. The target platform of the language was shared minicomputers operating in an interactive environment. The following sections describe the characteristics and implications of MUMPS as a programming language.

3.1.1.1 Interpretive Nature of MUMPS

MUMPS programs and source code are interpreted at run time. Source code instructions are converted into machine level instructions at the time of execution, rather than compiled prior to execution. The interpretive nature of MUMPS facilitates rapid prototyping, testing, debugging, and maintenance.

Rapid prototyping is facilitated because the program can be modified directly at run time. This saves the time required to compile a new version to determine whether it is acceptable.

EXHIBIT 3-1: DISTINCTIVE CHARACTERISTICS OF MUMPS

- MUMPS is an interpreted language
- Multiple, abbreviated statements can be included on a single line of program code
- Operator precedence in arithmetic expressions is ignored when parenthetical notation is not used, defaulting to strict left to right evaluation
- MUMPS supports a persistent, shared data base
- MUMPS variables are physically stored as variable-length character strings
- MUMPS does not support declaration statements
- Arrays are hierarchical and require no pre-allocation of memory
- Data on disk are stored in a sparse, hierarchical array
- MUMPS allows the use of string subscripts as keys to array nodes
- Temporary variable values are not removed from memory upon routine termination
- Shared data are available to other routines without program linkages or data declarations
- Data are physically stored in sorted order according to subscript values

Debugging is facilitated since the programmer can interact with the program at run-time and execute commands directly at the keyboard. The location of the error and the conditions that caused the error are immediately available.

Because MUMPS is an interpreted language, it is less efficient than a compiled language. In order to execute each command, MUMPS must first translate the source code instructions into machine instructions before executing them. Compiled programming languages perform this translation in advance and store the machine level instructions as a program file that is executed at run-time. Compiling programs improves efficiency since the translation of each statement occurs only once, not each

time it is executed; in addition, compilers optimize the instructions to perform faster execution.

In order to reduce the overhead associated with run-time interpretation, most implementations of MUMPS "tokenize" the code by partially compiling and optimizing the source code to improve execution speed. Tokenized MUMPS code still requires the MUMPS interpreter to operate. The dynamic structure of MUMPS at runtime makes it impossible to produce fully compiled MUMPS code because the compiler cannot predict the absolute memory addresses, required variable space, and instructions to be executed.

3.1.1.2 Language Syntax

The syntax of MUMPS programs is typically terse and cryptic. MUMPS makes extensive use of symbolic operators, as shown in exhibit 3-2.

EXHIBIT 3-2: MUMPS SYMBOLIC OPERATORS

The line of code shown below uses the indicated operators to tell MUMPS to check the variable SPONSORID for a sequence of 3 numeric digits, and if it finds them, to write the variable value on column 20, skip a line, with the variable inside the message "ID [sponsorid]: is valid.":

```
If SPONSORID?3N Write !,?20,"ID ",SPONSORID," : is valid."
```

The operators and their meanings are as follows:

| <u>operator</u> | <u>meaning</u> |
|-----------------|-------------------------------|
| ?3N | look for three numeric digits |
| ! | skip a line |
| ?20 | tab to column 20 |

MUMPS has a line orientation whereby several commands can be entered and executed from the same line of source code. Each line in a MUMPS program may have up to 255 characters. To support page printing, screen, and display width limitations, continuation of a line is indicated by three periods at the start of the next line. Any MUMPS command can be abbreviated to the first letter of the command. Exhibit 3-3 is an example of MUMPS code using one-character commands, two or more to a line.

EXHIBIT 3-3: ONE-CHARACTER COMMANDS IN MUMPS

The following MUMPS code fragment creates an array of numbers from one to 10, searches the array, and write the highest and lowest values encountered in the array:

```
For A=1:1:10 Set ARY(A)=A
S HI=0, LO=999999
F A=1:1:10 S:ARY(A)<LO LO=ARY(A) S:ARY(A)>HI HI=ARY(A)
W !,"Highest value = ",HI,!,"Lowest value = ",LO
```

Note the use of one-character command abbreviations:

S for Set
F for For
W for Write

Combining multiple abbreviated statements on a line makes code easier to write but harder to read. Many MUMPS developers establish programming conventions and standards to enhance the maintainability of the code through adequate program documentation and proper use of MUMPS syntax.

3.1.1.3 Routine Structure

The Portability Standard limits the collective size of MUMPS routines and local variables executing in main memory to 4,000 bytes. The actual allowable routine size is usually implementation specific. The allowable size of a routine depends on the operating system and hardware issues such as available memory, buffers, disk caches and other jobs.

MUMPS reads routines from disk or disk buffers when they are called to execute. Routines are removed from memory upon termination. Routines are swapped between main memory and disk storage as needed. The temporary variable space remains unchanged, allowing other routines access to temporary variables. Program linkage declarations are not supported, since routines in a program are not loaded in advance of execution.

The MUMPS routine structure has several implications for the developer:

- Local variables are available for other routines, eliminating the need to declare the variables explicitly.

- MUMPS does not automatically protect the programmer from unintentionally reusing local variables.
- The programmer must explicitly reinitialize local variables to prevent their unintentional reuse.

3.1.1.4 Declarative Statements

MUMPS does not support declarative statements for constants, data structures, variables, procedures, or functions. The following characteristics contribute to the lack of declarative statements in MUMPS:

- MUMPS is a command line oriented language. Declarations are defined at the time the function is required.
- MUMPS does not run in compiled mode, so linkages between separately compiled modules are not necessary.
- MUMPS variables and arrays are automatically available to all routines operating in the current process.
- MUMPS variables are physically stored as variable-length character strings.

The elimination of declarative statements in MUMPS is a major departure from other languages. Although the elimination of declarations may make programs easier to write, the use of declarations helps to define clearly the roles of variables and procedures in a program. Eliminating them can reduce maintainability, since declarative statements help to document a program.

3.1.2 MUMPS AS A DATA BASE MANAGEMENT SYSTEM

MUMPS was originally designed as both a general purpose programming language and a DBMS for Medical Information Systems. The principal function of a Medical Information System is to store, retrieve, and manipulate shared, textual data. Because of its integrated data base system, MUMPS stores and manipulates data differently from other programming languages.

3.1.2.1 File Structures

The MUMPS file structure is a sparse, hierarchical array implemented in a tree structure. MUMPS arrays are persistent; that is, they exist even after the routine that created them has stopped running. Local variables are shared by all applications using the same directory. MUMPS provides constructs that the programmer can use to prevent the re-use of local variables.

Most programming languages implement a sequential file structure in the form of a matrix. The matrix structure provides advantages in performing advanced statistical and mathematical functions. These file structures are usually static and hard to adapt to changing requirements. Selecting individual records from a MUMPS hierarchical file is very efficient compared to sequential matrix files.

3.1.2.2 Data Types

Most DBMSs support certain basic data types, such as integer and character, and abstract data types, such as date, time, and currency. MUMPS does not support any data types explicitly. All data are physically stored as variable length character strings. MUMPS automatically evaluates each variable according to the operation being performed. A numeric operation will return a numeric data type; a character operation, such as a concatenation, will return a character string type. MUMPS will evaluate a variable as either a character string, integer, floating point number, or Boolean value depending on how it is to be used.

Because numbers are handled as text strings, they must be converted into an internal numerical form that the computer can operate with directly. These conversions are not necessary for a language that supports integer and floating point numbers. Thus MUMPS is much less efficient at complex numerical calculations than a language that supports machine numbers. In addition, MUMPS requires additional space to store numbers since the packed machine level representations are not available.

MUMPS is essentially a non-typed language. A programmer has an unlimited set of operations that may be performed on data. However, data typing is useful in documenting the nature of an element, domains of acceptable values for the element, and logical operations that can be performed on the element.

3.1.2.3 MUMPS Arrays

Arrays (referred to in MUMPS as "globals") have no predetermined fixed or allocated size. Records in an array are variable length. Variable names may have up to thirty-

one character, allowing the programmer to describe the data completely; however, MUMPS uses only the first eight characters to distinguish between variables.

Most languages implement a matrix array structure with a predefined number of columns and rows. The variable length record structure of a MUMPS sparse hierarchical array makes efficient use of storage space. Disk storage requirements are determined by the amount of data physically stored, not by a pre-defined array size and record length; null values and redundant trailing spaces are not saved.

The branches in the hierarchical array structure are termed "subscripts". Subscripts are the keys to the data contained in each node in the array. Each subscript identifies a branch in the array, corresponding to a level in the array hierarchy. The physical ordering of data in an array is determined by the subscript values. Instead of storing data in the order they are entered, MUMPS stores data in sorted order according to the subscript values. Subscript values may be integers or character strings of up to thirty-one characters.

Storing in order by subscript values improves the search and retrieval ability of MUMPS. For example, a scan of records with the last name starting with the letter "G" will begin sequentially with the first record beginning with "G", terminating when it encounters a name beginning with the letter "H".

MUMPS arrays and subscripts are defined at the time a function uses the array. Exhibit 3-4 illustrates the hierarchical array structure of MUMPS.

The MUMPS hierarchical array structure preceded the relational table model. As in the relational model used in Relational Data Base Management Systems (RDBMSs), the data are normalized and redundant data are not stored within the system. Tables and attributes in RDBMSs are explicitly defined in advance of their use; in MUMPS, the "tables" and attributes are implicitly defined when they are used. The MUMPS sparse hierarchical structure is most useful for storing textual or other data whose records are of varying length. User views of the data in a RDBMS must be explicitly defined through the relationships between table values; however, hierarchical data bases implicitly contain relationship information that one record is dependent upon another.

Unlike the relational model, entity relationships are not established by matching values in key fields; instead, they are expressed through the hierarchy of the tree. This has the following implications:

- A relational data base is more flexible than a hierarchical data base, since tables can be related to each other in arbitrary ways, as opposed to only through dependency.

EXHIBIT 3-4: THE TREE STRUCTURE OF MUMPS GLOBALS ARRAYS

| <u>ARRAY</u> | <u>FIRST SUBSCRIPT (SSN)</u> | <u>FIRST SUBSCRIPT (DATE)</u> | <u>VALUE</u> |
|---------------|--------------------------------------|---------------------------------------|---------------------|
| PATREC | | | |
| | 372-40-1234 | | |
| | | | |
| | 372-40-1349 | 12/1/91 | "John Smith" |
| | | 1/1/92 | "Ultrasound" |
| | | 1/23/92 | |
| | 372-40-2299 | | |
| | | | |

The following statements assign the values shown in the boxes indicated in **bold** above:

```
Set ^PATREC(372-40-1349)="John Smith"
Set ^PATREC(372-40-1349,010192)="Ultrasound"
```


- A hierarchical data base is much faster to use for certain applications, since the records are implicitly related, and two sets of data do not have to be inspected and compared to locate matching values for each join.

Different user views of a MUMPS array would require the view to be defined at the physical level, storing an additional array with redundant subscripts and nodes. The inability to support different user views from the same physical structure is a significant drawback. If many user views are necessary, the storage efficiency of variable length record structures is largely negated by the additional redundant data.

3.2 MUMPS STANDARDIZATION

By the early 1970's, several dialects of MUMPS had evolved that were mutually incompatible. The widespread incompatibility severely hampered the portability of MUMPS applications between environments. The MDC was subsequently established to define a MUMPS Language Standard to be submitted to the American National Standards Institute (ANSI). The MDC is responsible for revisions to the MUMPS Language Standard and evaluates proposals to modify or enhance the language.

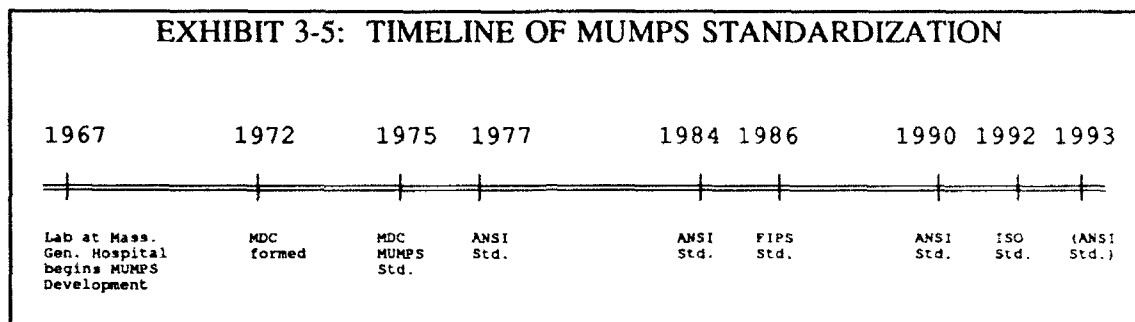
3.2.1 THE MUMPS DEVELOPMENT COMMITTEE

The MDC is composed of subcommittees that address general functional areas of the language, such as user interface or data management; each subcommittee is further organized into task groups which are responsible for a specific aspect of the language, such as X Windows or string handling. Any organization may request membership in the MDC status; so far, no membership requests have been denied.

Member organization representatives may also join task groups in which they have an interest. The MDC currently includes approximately 10 vendor and 50 user organizations. The member organizations send representatives to the three conferences that the MDC holds each year. Much of the conference time is spent working in task groups. A meeting of the full MDC is held at the conclusion of the conference. The Department of Veterans Affairs (VA) has provided a grant to pay for the annual meeting of the MDC; the DoD and Indian Health Service (IHS) have also periodically provided funds in the past. The MDC is also funded through annual membership dues of \$100.

The MDC has continued to assume the responsibility for maintaining and developing the MUMPS Language Standard. MUMPS has been approved as an ANSI Standard (ANSI/MDC X11.1-1990), a Federal Information Processing Standard (FIPS PUB

125), and an International Organization for Standardization/International Electrotechnical Commission Standard (ISO/IEC 11756). Exhibit 3-5 outlines the various stages of MUMPS development and standardization.



The MDC has published both a MUMPS Language Standard and a MUMPS Portability Standard. The ANSI Language Standard guides implementors on the minimal requirements that must be met in order to conform to the ANSI Standard. The Portability Standard is a subset of the ANSI Language Standard, serving implementors who create MUMPS interpreters and also developers who create MUMPS applications. Each standard is interpreted as a minimum specification to be met by implementors and a maximum specification to be exercised by developers. If both the implementor and developer adhere to the Portability Standard, the portability of the MUMPS program to other platforms will be assured. The existence of the Portability Standard has been a great benefit to MUMPS users. Both MUMPS developers and end users of MUMPS applications are able to take advantage of the higher performance and lower costs associated with newer technology.

Implementations of MUMPS may contain extensions to the ANSI Standard. Some implementation-specific extensions support operating system and hardware-dependent utilities. Other extensions to the current ANSI Standard include official "Type A Extensions" that the MDC has approved for inclusion in future standards in the process described below.

3.2.2 MUMPS STANDARDIZATION PROCESS

Anyone can submit proposals for extensions or modifications to the MUMPS Language Standard. The MDC is responsible for reviewing and approving all such proposed extensions or modifications. The MDC makes the ultimate determination as to whether or not the proposal is adopted as an extension or included in the next submission of the MUMPS Language Standard to ANSI.

3.2.2.1 MDC

Exhibit 3-6 shows the MDC's formal process for adopting modifications to the MUMPS Language Standard. The MDC is organized into subcommittees. Each subcommittee is responsible for several task groups. The MDC assigns requests for modifications to the appropriate subcommittee. The subcommittee is responsible for assigning the request to an appropriate task group or forming a new task group if an appropriate one does not already exist.

A request at this stage is termed a "Type C proposal." The proposal will be re-written in a formal manner and, if deemed worthy of further consideration, it is termed a "Type B proposal." When the proposal is deemed complete enough to be submitted to the full MDC for consideration it becomes a "Type A proposal." When a proposal receives type A status, no other changes to the proposal document will be accepted. If it is determined that changes are necessary, the proposal reverts to a type B proposal and is returned to the subcommittee for modification.

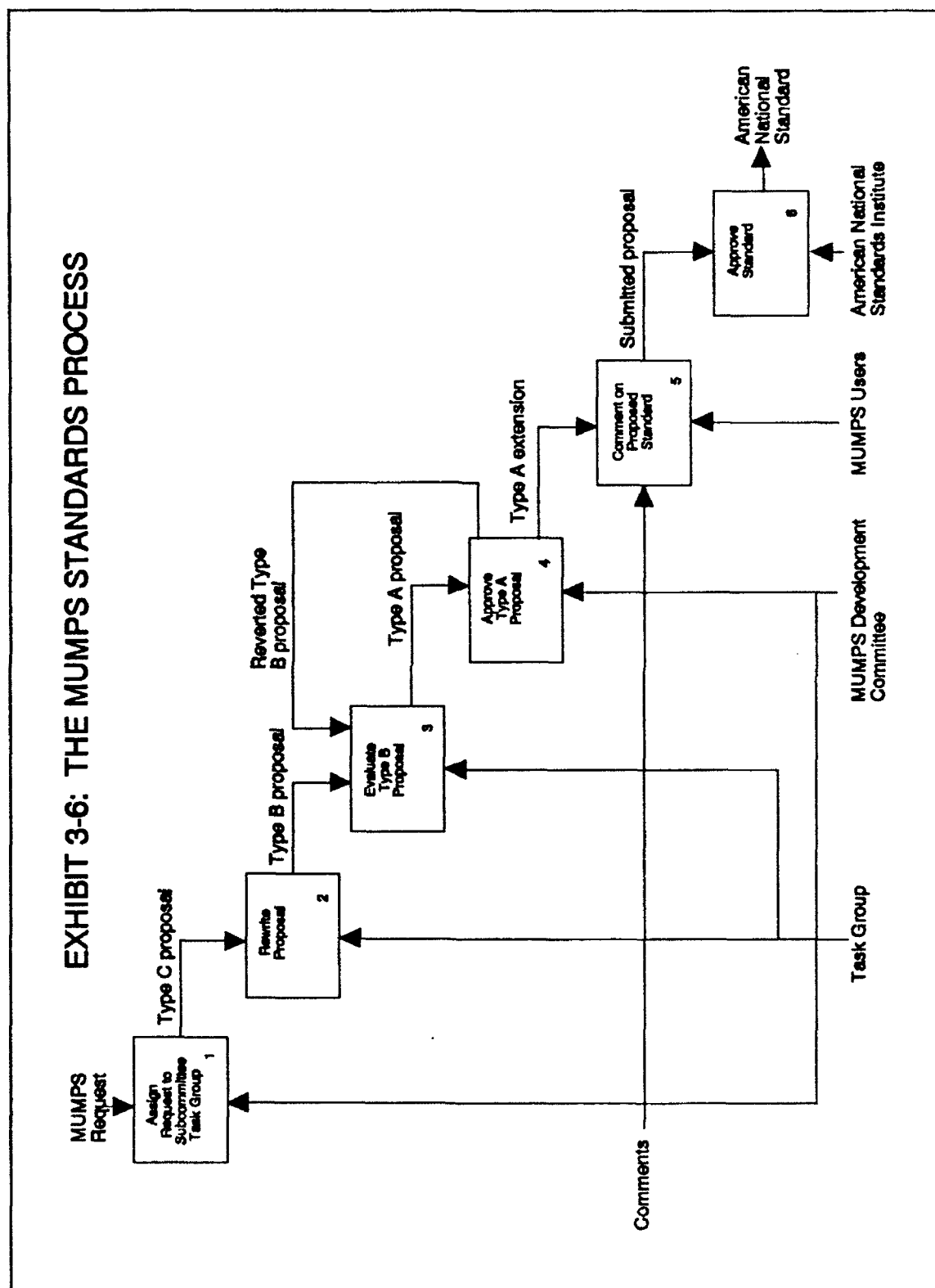
3.2.2.2 ANSI

Type A proposals that MDC approves receive "Type A Extension" status. All Type A Extensions are included in the next submission of the MUMPS Language Standard to ANSI. The MDC will seek comments from the MUMPS community on MDC Type A Extensions to the Language Standard. The comments received are submitted to ANSI along with the MDC MUMPS Language Standard. ANSI will approve the MDC MUMPS Language Standard when it is satisfied with the MDC responses to questions on specific proposals that arose out of the canvassing process. The main role of ANSI in the standardization process is to insure that the MDC has considered the material interests of all affected parties in preparing the MUMPS Language Standard.

The active involvement of the MDC in the continual improvement of the MUMPS Standard has played a significant role in the general portability of MUMPS applications. Proposed extensions to the Language Standard are widely publicized. The MDC actively seeks feedback from MUMPS users on proposed extensions to the ANSI Standard. This collaboration between the standards setting body and the user community has led to a MUMPS Language Standard to which both MUMPS implementors and application developers closely adhere. Vendor and user consensus over weaknesses in the ANSI Standard are included in future revisions to the standard.

Each subsequent revision to the ANSI Standard has addressed specific vendor and user concerns. Changes to the Language Standard that are not compatible with previous

EXHIBIT 3-6: THE MUMPS STANDARDS PROCESS



versions are adopted very cautiously. The most significant events and changes in the MUMPS Language Standard include:

- 1977 Standard -- Merged the several incompatible dialects of MUMPS that had evolved into a single standard
- 1984 Standard -- Implemented the use of descriptive subscripts in MUMPS arrays
- 1990 Standard -- Provided support for structured programming techniques
- 1993 Standard (proposed) -- Will provide support of MUMPS bindings to industry standards (SQL, X-Windows, GKS)

3.2.2.3 NIST

NIST follows a standard process in recommending to the Secretary of Commerce the establishment or revision of a standard. A sponsoring organization, in this case the MDC, submits a standard to NIST for consideration. NIST elicits comments on the standard from interested or affected parties. Comments are accepted for approximately three months. NIST then reviews comments and prepares responses to them. The sponsoring organization may provide technical background to support NIST's response to the comments. When all comments have been addressed to NIST's satisfaction, the standards will be forwarded to the Secretary of Commerce for approval.

After the approval of the Secretary of Commerce, NIST issues a FIPS PUB specifying the standard. The FIPS PUB becomes effective six months after the date of approval. A twelve month implementation period commences on the effective date of the standard. This implementation period gives implementors and developers time to adjust to the adoption of the new standard. At the end of the implementation period, government usage of the standard becomes mandatory.

As is common, FIPS PUB 125 follows for MUMPS the form of the current ANSI. It is not uncommon for a FIPS PUB to be superseded by a recently revised ANSI. In this case, the FIPS PUB and the ANSI are different until the FIPS PUB is revised.

3.2.3 1977 ANSI Standard

The 1977 ANSI Standard was based on the initial work of the MDC. The ability of the MDC to gain the consensus of the MUMPS community in consolidating the numerous incompatible dialects of MUMPS into a single standard was a significant

accomplishment. The MDC established a formal process of standardizing the language that considered the interests of the MUMPS community. In 1977 ANSI formally verified this process with the approval of the MDC MUMPS standard.

4.0 CURRENT FIPS PUB

The current MUMPS FIPS PUB, FIPS PUB 125, is based on the previous MUMPS ANS, ANSI/MDC X11.1-1984. FIPS PUB 125 adds introductory text and an appendix to the 1984 ANS. This chapter relates FIPS PUB 125 to the AST criteria.

4.1 1984 ANSI STANDARD

The 1984 revision of the ANSI Standard added and extended to the MUMPS Language Standard. The following are some of the more important changes:

- String subscripts
- New \$Order function to get the next subscript at the current level
- Improved \$Piece function to allow an individual field to be set
- String indirection
- Read enhanced

Each of these is described below.

4.1.1 STRING SUBSCRIPTS ADDED

The 1977 ANS allowed only positive integers as the subscripts of arrays. In the 1984 ANS, valid subscript types were expanded to include negative integers, real numbers, and text strings. This enhancement allowed subscripts to contain descriptive information. Exhibit 4-1 gives an example of the use of string subscripts in an array. Each subscript represents a different level in the array hierarchy. Multiple subscripts are also allowed. The 1984 ANS also included support for the searching of arrays with string subscripts.

4.1.2 \$ORDER FUNCTION ADDED

The \$Order function added a way to search MUMPS arrays to locate data. \$Order returns the next subscript in the array, or a conventional null value if there are no more items in the array. Exhibit 4-2 gives an example using \$Order.

The \$Order function replaced the \$Next command, which performed a similar function for positive numerical subscripts. \$Next would give an incorrect result if a string subscript or negative subscript was used; the \$Order function provides the correct response.

EXHIBIT 4-1: EXAMPLE OF STRING SUBSCRIPTS

Suppose that the following variables have the values shown:

| <u>variable</u> | <u>value</u> |
|-----------------|--------------|
| PATIENT_SSN | 372-40-1349 |
| DATE | 1/1/90 |
| CLINIC_ID | Radiology |

Suppose further that the user enters "MRI-PATELLA" when the read statement below is executed:

```
READ ! "Enter type of service: ", PROC
SET SERVICE(PATIENT_SSN, DATE, CLINIC_ID)=PROC
```

After these two statements have been executed, the value for SERVICE("372-40-1349", "1/1/90", "Radiology") is "MRI-PATELLA".

EXHIBIT 4-2: EXAMPLE OF \$ORDER FUNCTION

Suppose that the array THING has the following definition:

```
THING
├── 'ANIMAL'
├── 'MINERAL'
└── 'VEGETABLE'
```

Then \$Order(THING('ANIMAL')) is 'MINERAL' since MINERAL is the next subscript after ANIMAL.

4.1.3 \$PIECE FUNCTION ENHANCED

In the 1977 Standard, the \$Piece function was used to extract specified substrings from strings. The 1984 ANS enhanced the \$Piece function to allow a programmer to assign a value to a portion of a data string. Assignment could be done in place, whereas previously, the string had to be exploded and concatenated back together.

4.1.4 STRING SUBSCRIPT INDIRECTION ADDED

String subscript indirection provides support for general data access. The name of a variable and its subscripts can be evaluated at the time they are used. Indirection

allows generic array handling routines to be written. For example, exhibit 4-3 is a code fragment from a routine that can display the contents of any variable.

EXHIBIT 4-3: EXAMPLE OF SUBSCRIPT INDIRECTION

Suppose the following variables have the values shown:

| <u>variable</u> | <u>value</u> |
|-----------------|---------------|
| PATREC(1) | "918-39-9381" |
| VAR | PATREC |
| SUBS | 1 |

The following code segment displays the data for the given variable and subscripts using subscript indirection:

```
Write "Data = ",@VAR@(SUBS)
```

The above code segment would display "Data = 918-39-9381" when it is executed.

4.1.5 READ COMMAND ENHANCED

The Read command was enhanced to allow the programmer to specify a limit to how many characters can be read into a variable. For example, when reading a state abbreviation, the programmer can specify to read no more than 2 characters, even if the user enters more characters.

4.2 DIFFERENCES BETWEEN FIPS PUB AND 1984 ANS STANDARD

The current FIPS PUB is the same as the 1984 ANS, with two exceptions. The FIPS PUB adds a foreword, which provides a short history of MUMPS. The FIPS PUB also includes a table of the 128 ASCII characters and their decimal values which determine the collating sequence of stored data. MUMPS data are collated as follows:

- empty strings (i.e. '')
- "canonical" numbers (e.g. 2) in increasing order
- strings (e.g. 'box 12') in increasing ASCII decimal value

In MUMPS, 'ABC' follows '', 12, 'AB', and 'ABB'. A character with a larger ASCII decimal value follows a character with a smaller ASCII decimal value.

The purpose of including this table in the FIPS PUB is to enhance portability by ensuring that all standard versions of MUMPS use the same collating sequence.

5.0 CURRENT MUMPS STANDARD AND PLANNED FIPS PUB

The current MUMPS Language Standard was approved by ANSI in 1990 (ANSI/MDC X11.1 - 1990). The 1990 ANS introduced the following changes:

- Structured programming techniques to improve the readability and flow control over MUMPS programs
- Support for libraries of functions through the introduction of variable scoping within routines and parameter passing to routines
- A translation function to change characters in a variable to user specified criteria, such as converting characters from lower to upper case
- Improved formatting and output of numeric data
- Improved functions to query and retrieve data, simplifying programming

The MDC has submitted a new MUMPS language standard to NIST for a revised MUMPS FIPS PUB that is based on the current 1990 ANS. Although these new features are all part of the proposal, it is possible that NIST could remove one or more features from the FIPS PUB if issues resulting from the comment stage of the review process are not resolved to the satisfaction of NIST or the Department of Commerce. At this time NIST expects that the new FIPS PUB will follow the 1990 ANS. The 1990 ANS improves the completeness of the MUMPS language.

The following discussion describes the 1990 ANS and proposed changes to the MUMPS FIPS PUB in the context of the AST criteria. This discussion is based on the assumption that the language features defined in the 1990 ANS will be adopted by the MUMPS FIPS PUB.

5.1 BLOCK STRUCTURING

The 1990 ANS added the execution of nested blocks of statements. This eliminated the need for non-structured branching in code. Previously label references were required to conditionally execute a set of statements in a program.

Exhibit 5-1 shows two versions of a MUMPS program, one coded according to the 1984 standard, and the other using features of the 1990 standard. Program flow is easier to follow in the second version, since it does not jump back and forth between

label references. The support of more structured programming techniques provides a significant improvement in the readability and maintainability of MUMPS programs.

Block structuring in the proposed FIPS PUB will provide improvements with respect to AST maintainability. The readability of source code will be enhanced when the use of "Go To" instructions are limited as a result of block structuring.

EXHIBIT 5-1: EXAMPLE OF BLOCK STRUCTURING

The following MUMPS program conditionally executes a set of statements by referencing labels in the program, in a style typical of code that adhered to the 1984 Standard (text following a semicolon ";" is a comment):

```
PTN      ;Records patient allergy and ailment information prior to
        ;the performance of any services in a clinic or hospital.
        Read !,"Is the patient allergic to any medication (Y/N)?"
        If ANS["Y" Do MEDCTN
        Do AILMNT
MEDTCN   For N=1:1 Read !,"List medication:",MEDIC Quit:MEDIC=""
        ...Read !,"Describe reaction:",REACT
AILMNT   For N=1:1 Read !,"Describe ailments:",AIL Quit:AIL=""
        ...Read !,"Enter date of ailment:",DATEAIL
        Quit
```

The flow of the same program coded with block structuring would be easier to follow:

```
PTN      ;Records patient allergy and ailment information prior to
        ;the performance of any services in a clinic or hospital.
        Read !,"Is the patient allergic to any medication (Y/N)?"
        If ANS["Y" Do
        . For N=1:1 Read !,"List medication:",MEDIC Quit:MEDIC=""
        . ...Read !,"Describe reaction:",REACT
        For N=1:1 Read !,"Describe ailments:",AIL Quit:AIL=""
        ...Read !,"Enter date of ailment:",DATEAIL
        Quit
```

5.2 VARIABLE SCOPING

Commands and functions were added to support the creation of function and routine libraries that produce results independent of the contents of variables and functions that are not being used. These commands and functions allow variables to be redefined to shield other variables. Parameters could be passed to routines.

The following is a description of these commands and functions included in the 1990 ANS. These changes in the 1990 ANS will increase AST usability, productivity, and maintainability in the new FIPS PUB.

5.2.1 NEW COMMAND

The New command was added to allow the programmer to redefine variables within a single routine. This command allows the programmer to prevent accidental "side effects" that unintentionally change local or global variables used by other routines. The New command has two forms: one to redefine a specific set of variables, the other to redefine all but the specified variables. Exhibit 5-2 provides an example of the use of the New command.

EXHIBIT 5-2: EXAMPLE OF NEW COMMAND SYNTAX

The following example illustrates the effect of the New command in a MUMPS program.

```
MAIN      ;Illustrates the use of the new command
          Set X=100,Y=-10
          Do CALC
          Write !,"X=",X
          ;
CALC      New X,Y
          Set X=0,Y=50
          Write "X=", X, ", Y=",Y
```

The screen would display the following results from the program:

```
X=0, Y=50      (from CALC routine)
X=100, Y=-10   (from MAIN routine)
```

After the redefinition and output of X and Y in the CALC routine, the variables X and Y are returned to their original values set in the calling program, 100 and -10.

5.2.2 PARAMETER PASSING

The 1990 ANS added the ability for pass parameters between routines. This allowed variables and entire arrays to be transferred from one routine to another. An implied New command is executed on the parameter list in the subroutine (referred to as the formal list). Parameters are passed from a calling routine (referred to as the actual

list) to a sub-routine based on the relative position of parameters in the actual and formal lists. MUMPS does not allow non-blank spaces in the actual list.

Parameter passing improves the ability of a programmer to develop shared routines that perform a specific function. Utility functions could be more easily separated from core program code, reducing the duplication of program code. Exhibit 5-3 illustrates the use of parameter passing in a MUMPS program.

EXHIBIT 5-3: EXAMPLE OF PARAMETER PASSING SYNTAX

The following is an example of parameter passing in a MUMPS program. The subroutine PVALUE calculates the present value of a lump sum payment for 1 interest period.

```
PVALUE(I,P,PV)
    ;Calculate present value of lump sum. I=interest
    ;P=Principal, PV=Present value. Assumes that
    ;calculation is for one interest period.
    Set PV=P/(1+I)
    Write ! "Present value is: ",PV
```

Calling the program PVALUE with the Do statement....

```
Do PVALUE(.08,1000)
```

...yields the following results

```
Present value is: $925.93
```

The values .08,1000 in the actual list are passed by position to the variables I and P in the formal list. An explicit New function is performed on the variables in the formal list (I,P,PV). The variable PV is undefined since it does not have a corresponding position in the actual list.

5.2.3 EXTRINSIC FUNCTIONS

The 1990 Standard added the ability to call extrinsic functions, i.e., functions that are not defined as part of the MUMPS language standard. Implementors commonly add extrinsic functions in their products. Extrinsic functions can return a single value; these provide support for advanced mathematical functions, such as trigonometry. The syntax of an extrinsic function precedes the function name with two dollar signs ('\$\$'). These functions could be executed by implementors in more efficient ways

that take advantage of hardware arithmetical capabilities. Exhibit 5-4 provides an example of an extrinsic function call.

EXHIBIT 5-4: EXAMPLE OF EXTRINSIC FUNCTION CALL

The following is an example of the syntax for calls to extrinsic functions. Unlike the routine PVALUE in Exhibit 5-3, the extrinsic function \$\$PVALUE in this example performs present value calculations that span several interest periods.

```
Set I=.0067,P=1000,N=12  
PV=$$PVALUE(I,P,N)
```

The extrinsic function will return the value for PV to the calling program.

5.3 FORMATTING FUNCTIONS

The 1990 Standard added functions to format strings. The \$Translate function replaces character sets with other characters. The \$Fnumber function allows numbers to be formatted according to the programmer's conventions.

AST usability, productivity, and maintainability will be improved in the new FIPS PUB for program code requiring character translations and numeric output. The FIPS PUB will be more complete with respect to string operations.

5.3.1 \$TRANSLATE FUNCTION

The \$Translate function in the 1990 ANS will simplify the programming code required for performing character translations. The \$Translate function converts sets of characters found in the input string to other sets of characters. This could be used to convert characters to upper case, delete blanks, and reformat characters. Exhibit 5-5 illustrates the use of translation function.

5.3.2 \$FNUMBER FUNCTION

The \$Fnumber function allows numbers to be formatted with commas, force plus signs to print, place parenthesis around negative numbers to be parenthesized, suppress minus signs, and switch from leading signs to trailing signs. The \$Fnumber function provides better support for financial and statistical type applications and makes the language more flexible. The \$Fnumber function will simplify the program code necessary to represent numeric data in the user's desired format. Exhibit 5-6

EXHIBIT 5-5: EXAMPLE OF TRANSLATE FUNCTION

The translation function has the form of \$Translate (Expression,Replace,With). The following is an example of the use of the translation function which converts upper case letters to lower case.

```
Set Upper="ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
Set Lower="abcdefghijklmnopqrstuvwxyz"  
Set X="department of health affairs"  
Write ! "Translation = ", $Translate(X, Lower, Upper)
```

The result of these statements would be:

```
Translation = DEPARTMENT OF HEALTH AFFAIRS
```

illustrates the use of the \$Fnumber function.

EXHIBIT 5-6: EXAMPLE OF \$FNUMBER FUNCTION

The following is an example of the \$Fnumber function to insert commas into the appropriate position in the number:

```
Set X=13254343  
Write $FN(X,""); result should be 13,254,343
```

5.4 SEARCH FUNCTIONS

More powerful functions to retrieve data and query data were added to the language in the 1990 ANS. In the current FIPS PUB a validity check must be performed to determine if the variable contains a valid value, requiring more complex code to traverse a MUMPS database. The \$Get function and the \$Query functions proposed in the new FIPS PUB will provide easier and more efficient ways of extracting information from variables.

Fewer lines of program code required to perform these functions specified in the 1990 ANS will increase AST productivity and maintainability.

5.4.1 \$GET FUNCTION

The \$Get function retrieves a value from a variable or array. The \$Get function also incorporates a check to determine whether the variable exists or is undefined. Instead of causing an error when the variable does not exist, \$Get returns an empty string. Exhibit 5-7 illustrates the use of the \$Get function.

EXHIBIT 5-7: EXAMPLE OF \$GET FUNCTION

Prior to the 1990 ANSI standard, the following syntax would be required to retrieve a variable value from the global ^Benef using the \$Data function. PatID and Patname are subscripts in the array.

```
PDAT      ;If $Data evaluates to true store results in Pdata.  If not
           ;true, output message.
           If $Data(^Ben(PatID,Patname))  Set Pdata=^Ben(PatID,Patname)
           Else Set Pdata=""
           If Pdata="" Write ! "Patient name not found."
```

If the node did not exist, the \$Data function would return an error. When using the \$Get function it is not necessary to know in advance whether or not the node exists. The above could be re-written as follows:

```
PDAT      ;Same as above, but using $Get function
           Set Pdata=$Get(^Benef(PatID,Patname))
           If Pdata="" Write ! "Patient name not found."
```

5.4.2 \$QUERY FUNCTION

The \$Query function provides a way to retrieve the subscripts of the next node with a defined data value. This makes tree traversal much easier, since empty nodes in the tree are not returned; \$Query incorporates tests to determine whether data exists or is defined. The \$Query function returns the entire reference for the next data node containing a value, regardless of the level. The reference returned can subsequently be used to retrieve the value in the node. With the \$Query function programmers can manipulate the variable data without knowing the specific structure of the data base.

5.5 MUMPS FIPS PUB STATUS

Currently, the new FIPS PUB is in the comment stage. The comment stage will be completed by mid-November 1992. The MDC will supply NIST with the MUMPS technical background to support their responses to the comments. NIST estimates that the new standard may be proposed to the Secretary of the Department of Commerce in January 1993. The new MUMPS FIPS PUB will be effective six months after approval by the Department of Commerce, whose approval is expected in January or February. A twelve month implementation period will commence once the standard is effective. During this period use of the new MUMPS standard by Federal Agencies is not mandated. Use of the standard will become mandatory eighteen months after the initial Department of Commerce approval.

6.0 PROPOSED 1993 ANS

The MDC is currently preparing a revised MUMPS Language Standard to be submitted to ANSI. The proposed 1993 ANS standard contains many significant changes relevant to DoD's effort to identify and use products supporting industry standards. An awareness of the proposed changes to the ANS standard will provide insight into the changes that could be expected in future MUMPS FIPS PUBS.

6.1 PROPOSED CHANGES TO MDC MUMPS LANGUAGE STANDARD

The MDC has addressed several areas of the language in the new MDC MUMPS Language Standard to be proposed to ANSI. The most significant additions have occurred in interfaces to industry standards, transaction processing, error processing, and networking/remote system access. These changes improve the flexibility, completeness, and stability of the MUMPS language.

The following sections discuss these areas and other proposed changes and additions to the ANS standard in the context of the AST criteria.

6.1.1 INTERFACES TO INDUSTRY STANDARDS

There has been a great deal of interest in the MUMPS community for improved support for interfaces both to industry standards such as SQL, X-Windows, and GKS, and to other programming environments. The proposed ANS standard includes several additions that make significant progress in interfacing MUMPS with other environments.

6.1.1.1 External Program Calls

The current methods of calling external programs from within a MUMPS program are vendor specific. Vendor specific functions added to the language are referred to as "Z" functions and must be prefaced with a "\$Z". Since Z functions are vendor specific, they may not be portable between implementations.

The proposed ANS standard specifies a standard method of accessing programs external to MUMPS, such as C subroutines, SAS and Fortran programs for advanced statistical and mathematical functions, etc. External callable routines can return a value to the MUMPS program or simply perform a function external to the MUMPS program.

This capability will increase AST productivity and portability by standardizing the method of calls to external programs and improving the flexibility of the language. The instructions in a called program are in a 'black box', not directly accessible by a MUMPS programmer. Hidden source code may decrease AST maintainability, but overall AST maintainability is more likely to be increased by allowing more reuse of programs from a standard library. Exhibit 6-1 illustrates the syntax for calling external programs that has been proposed for the ANS standard.

EXHIBIT 6-1: CALLING EXTERNAL ROUTINES FROM MUMPS

The proposed MUMPS syntax for calling external routines consists of four elements:

- An ampersand (&) to indicate that the routine or program is external to MUMPS
- The name of the package that provides a binding between MUMPS and the external routine (to be assigned by MDC or implementors)
- A period (.) to separate the package name from the routine
- The name of the external routine or program and its parameters

The following is an example of how a program might call an external routine in an X-Windows system:

```
Set Display=$&XLIB.XOpenDisplay(MYNODE)
```

This method of external routine calling allows MUMPS to integrate with bindings to other languages, interfaces, and other external program libraries. The only portion in the above statement that needs to be defined within MUMPS is the package name "\$&XLIB". XLIB is the name of the package that contains libraries of standard routines with X-Windows and defines the external routine structure.

6.1.1.2 X-Windows Binding

The proposed ANS standard includes a binding to the X-Windows graphical user interface system. FIPS PUB 158, based on MIT's X-Windows system, provides specifications on client-server graphical user interface processes. MUMPS currently lacks support for the windows, dialog boxes, and menu bars that an X-Windows

binding would provide. The proposed standard will establish an X-Windows binding with X-Windows libraries Xlib, Xtoolkit, Xmotif, Xmotifm, and Xmumps. These libraries would be accessed using the new MUMPS external calling syntax, shown in exhibit 6-1.

The X-Windows binding provides MUMPS developers with the ability to develop MUMPS programs that use the graphical interface services available in the X-Windows system. Standardizing the binding to X-windows will improve AST portability by eliminating the use of implementation specific X-windows bindings. The flexibility of the language will be improved to support applications requiring graphical interface services.

6.1.1.3 SQL Binding [reference SQL standard below].

The ability to embed Structured Query Language (SQL) syntax in a MUMPS program is specified in the proposed ANS standard, via the \$SQL function. The SQL binding will allow the retrieval and manipulation of data in a MUMPS hierarchical array through both embedded SQL query statements and MUMPS routine calls from within an SQL program. The binding is specified in the SQL standard. Although several vendors have integrated MUMPS/SQL products, the integration is vendor-specific and generally is not portable to other vendor products.

Embedded SQL commands will over-ride the MUMPS 255 character line limitation. The MUMPS interpreter will accept interactive and embedded SQL commands from users, allowing the MUMPS system to function as a SQL server. Embedded syntax from other languages such as C, Pascal, Fortran, and Ada may be supported in future MUMPS standards in a similar fashion.

Embedded SQL commands will improve AST portability, usability, and productivity. Non-procedural SQL instructions may be used to retrieve and manipulate data. The flexibility and completeness of the language is improved with the proposed SQL binding.

6.1.1.4 Graphical Kernel System

A binding to the Graphical Kernel System (GKS) is included in the proposed ANS standard to address the graphical limitations of the MUMPS language. Currently, MUMPS supports only text output; the GKS binding would add 2-dimensional graphics to MUMPS. In addition, GKS is an open systems technology. The VA has been very active in the GKS binding by providing funding for the testing and

development of the binding at the University of Lowell's Graphics Research Laboratory.

The specifications for the GKS binding will be finalized shortly. The method of the GKS binding will be similar to the method employed in the X-Windows binding. AST portability and usability will be increased with the standardization of a GKS binding. The completeness and flexibility of the language will be improved with graphics support.

6.1.2 TRANSACTION PROCESSING SUPPORT

The ability of MUMPS to support transaction processing standards is important for its acceptance as a general purpose business language. The proposed ANS standard supports the ACID (Atomic, Consistent, Isolated, and Durable) test for transaction processing data base updates. These four elements are defined as the following:

- Atomicity -- data base updates must occur at the transaction (unit) level; partial updates upon error conditions are not acceptable.
- Consistency -- transactions must result in a consistent data base at all times.
- Isolation -- transactions must be isolated from each other; transactions must produce the same result whether running concurrently or sequentially.
- Durability -- Once a transaction has been reflected in the data base, it must continue in that state regardless of system failures.

Several new commands are specified in the ANS standard to maintain data base integrity. These commands, described below, reduce the risk of data base corruption by ensuring that data base updates occur as a unit or not at all. The capitalized portion of the command refers to the MUMPS abbreviated syntax for the command.

- TStart will indicate the start of a transaction unit and the local variables to be reset in the event of a restart. The variables defined become part of the transaction unit, reducing the chance of partial data base updates.
- TCommit will perform the posting function for the transaction. TCommit will terminate the transaction and make permanent and available all data base changes.
- TRestart will restart a transaction, resetting all variables identified in the TStart command.

- TROlback will terminate a transaction and reverse all data base updates that occurred during the transaction.

The proposed transaction processing support in the ANS standard will increase AST usability and productivity for applications requiring these features. The flexibility and completeness of the language will be improved with the inclusion of these commands in the language standard.

6.1.3 STANDARDIZED ERROR PROCESSING AND HANDLING

Standardized methods in dealing with program error conditions are included in the proposed ANS standard. Current implementations of MUMPS make use of implementation specific "Z" functions to address error handling. The standardization of error handling increases AST portability and maintainability. Previously, many of these features were incorporated into vendor products as implementation specific extensions to the language. The proposal will standardize several areas of error processing, which are described below.

6.1.3.1 Standard Error Identification

Standard error codes will be specified by the MDC and incorporated into the ANS standard. Each standard error code will be prefaced with the letter "M". Implementors will be able to add their own error codes and error messages in their products. Error codes will be stored in a special variable in comma delimited strings. At the start of a program, the value of this variable will be a null. Program code will check the value of this variable. When the value is null, program execution continues in its normal manner. If the variable contains an error code, error processing instructions will assume control of the program.

6.1.3.2 Program Defined Transfer of Control

Programmers will be able to specify instructions that are to be executed when an error condition is encountered.

6.1.3.3 Identification of Program Code at Time of Error

New variables will be introduced that provide information about how the program arrived at its current state or the state of the program during the last error condition. This will aid the programmer in identifying the code contributing to the error.

6.1.4 STANDARDIZED DEVICE HANDLING

Currently program code necessary to interface with physical devices may be implementation specific. Parameters that are necessary to control devices are not currently standardized. The proposed ANS standard provides specifications for standardized device parameters, minimizing the need for developers to use implementation specific code.

New variables will be introduced that provide information about the original device, current device, and control sequences that terminated the last read command. Device synchronization will be improved by allowing the resetting of system variables that report the current cursor location on the screen.

The standardization and improvements in device handling will minimize implementation specific code and improve programmer control over device synchronization, resulting in increased AST portability and usability.

6.1.5 IMPROVED NETWORKING, REMOTE ENVIRONMENT SUPPORT

Several specifications are included in the proposed ANS standard to improve the ability of MUMPS to integrate data across networks and systems. These specifications will also serve to improve the interoperability between different MUMPS implementations.

The ability to access and share remote routines and integrate data across dispersed systems will increase AST usability, productivity and maintenance. The flexibility and extendibility of the language will be improved.

6.1.5.1 Open MUMPS Interconnect

The Open MUMPS Interconnect (OMI) is a proposed system level protocol that will allow information on different machines or implementations to be shared. The OMI protocol will allow this to be accomplished transparent to the MUMPS code.

6.1.5.2 Remote Global and Routine Library Access

The proposed ANS standard will allow arrays and routines that do not reside on the machine or implementation to be referenced in standard MUMPS statements by prefacing the statement with the environment name. This will allow the execution of

remote routines and the setting of local or global variables to the values in a remote variable.

6.1.5.3 Unique System Identification

A new variable \$SYSTEM will be introduced that provides an identification of the current MUMPS execution. All jobs in process will be unique within this identification. The \$SYSTEM variable will allow a process to identify itself and its data uniquely, regardless of where it is located or connected.

6.1.6 IMPROVED DATA ACCESS AND MANIPULATION

The new ANS standard includes new functions and commands as well as extensions to existing functions and commands to improve access to data in a MUMPS database and the manipulation of data.

The new functions and commands, described below, will increase AST usability, productivity, and maintenance. Fewer lines of program code will be necessary to perform data access and manipulation functions. The language standard will be more complete in these areas.

6.1.6.1 Data Base Manipulation

A new command, Merge, is included in the proposed ANS standard to copy or move the contents of a subtree of a MUMPS data base to another subtree. This eliminates the need for program looping operations to locate all of the defined values in a subtree and set them to values in another subtree. This is a very common function when editing records, where the program makes a temporary copy of the data, edits the copy, and then copies it back into its original location.

6.1.6.2 Global Variable References

The proposed ANS standard includes an additional function (\$Name) that returns the reference name of a subscripted global or local variable (the name that would be used to set or retrieve the variable). This function simplifies code where the name of a subscripted value must be passed to functions or routines.

6.1.6.3 Character string manipulation

A new function, \$Reverse, is introduced that will reverse the order of a character string.

6.1.6.4 Null or Undefined Substrings

The \$Get function, which is used to retrieve a variable value, will be extended to indicate whether a storage location is defined for the variable reference. In this manner, if a null value is returned the system will indicate if the null value is contained in an existing node or if the null value is the result of an undefined storage location for the variable.

6.1.6.5 Searching in Reverse Order

The \$Order function is used to search through MUMPS hierarchical arrays and retrieve all defined subscripts at the specified level. Since data are stored in sorted order, values are returned in this order. The extension to \$Order will provide the ability to specify that subscripts be searched in reverse order (eg. Z-A).

6.1.6.6 Direct Input to Global Variables

The read statement has been extended to include global variables as the destination. Currently data read from an input device can only be read into a temporary (local) variable. The extension will allow the data to be read directly into a global variable.

6.1.6.7 Iterative Pattern Matching

Pattern matching will be extended to include pattern elements composed of several patterns. Matches on the different patterns can be evaluated based on the use of logical OR's and AND's. This extension allows checks for different patterns to be included in the same statement. Extended pattern matching could be used to parse social security numbers or dates, for example.

6.1.7 EXPONENTIATION OPERATOR

An exponentiation operator (**) is included in the proposed ANS standard, improving the ability of MUMPS to perform operations on numeric data, particularly those

involving financial calculations such as present and future value annuity calculations, amortization schedules, etc.

AST usability, productivity, and maintenance will be improved with the addition of an exponentiation operator. The flexibility and completeness of the language will be improved to support current and new applications.

6.1.8 IMPROVED PARAMETER PASSING

Parameters are passed between MUMPS routines based on the relative position of the parameters in the parameter list of the calling routine (referred to as the actual list) and the parameter list of the subroutine (referred to as the formal list). Passing parameters between a calling routine and subroutine helps ensure that variable names defined by the calling routine are not unintentionally re-used. The parameters passed from a calling routine may be actual values or references to local variables or arrays. The proposed ANS standard will improve parameter passing by preventing duplicate variable names in the formal list. Empty positions will be allowed in actual list. Allowing empty positions makes it easier to include undefined parameters in a subroutine, providing higher generality and flexibility. This is a common feature in other languages. Currently, undefined parameters in the formal list must appear at the end of the list, where no corresponding position in the actual list exists.

The proposed changes to parameter passing will increase AST usability and maintainability. A more conventional method of handling undefined parameters will be available to programmers. The risk of program failures will be reduced with the elimination of duplicate variable names.

6.2 PROPOSED CHANGES TO PORTABILITY STANDARD

Several changes are proposed in the Portability Standard to extend current limits specified for portable MUMPS programs. These proposed changes will provide a programmer more flexibility in developing applications while insuring portability across different MUMPS implementations. Implementations of MUMPS at a minimum must meet the new specifications proposed in the portability standard. The following are some of the key portability changes:

- In block structured programs, the number of nesting levels allowed for portable applications will be increased from 30 to 127.
- The number of significant digits for numerics or calculations will be increased from 12 to 15 digits.

- The length of an individual subscript will be increased from 63 characters to the limit on the variable reference length.
- The reference length for a variable (the total length of all subscripts) has been increased from 127 to the allowable program line length, currently 255 characters.

Applications that meet the proposed limits in the standard but exceed the current limits will become portable when all vendor products support the proposed limits. Additional flexibility will be available in developing portable applications with the improvements in the standard. These changes in the portability standard will increase AST usability and portability.

6.3 STATUS OF MDC MUMPS LANGUAGE STANDARD

The proposed 1993 ANS standard is scheduled to be finalized at the February meeting of the MDC. Although the MDC refers to the proposal as the "1993 ANS standard", it is very likely that ANSI approval of the standard will not be received until early or mid 1994.

The MDC and ANSI have already agreed on a list of individuals to be canvassed. Submission of the proposal to ANSI and to the canvass list for comments is planned for the summer of 1993. Approval of the proposal by ANSI could occur in late 1993.

The significance of the changes in the MUMPS standard, particularly those that address methods of supporting industry standards, may result in delays during the canvass process. It is possible that ANSI will request a second canvass based on comments received in the first canvass. A second canvass could delay ANSI approval of the standard for 4 or 5 months, or until mid 1994.

The MUMPS language standard is becoming more complete, with the flexibility to support a wider range of applications through language features and access to routines in other languages. The ANS standard itself will become more stable as the MDC standardizes the methods employed to integrate MUMPS with industry standards. As the language matures future ANS standards of the language should prove to be less of a moving target for developers and implementors.

7.0 SUMMARY

This chapter summarizes the changes being made to the MUMPS FIPS PUB and programming language with respect to AST capabilities. Information about the planned changes to the MUMPS FIPS PUB was gathered through interviews with staff of NIST. MDC members and contacts provided by MUG were consulted to gather information about the changes planned for the MUMPS programming language.

FIPS PUB 125 is being revised to reflect the current Standard. The MUMPS language standard is also being revised for 1993. Both revisions are in the final comment stage and will be finalized in 1993 or early 1994. The revisions to FIPS PUB 125 and the MUMPS' ANS will improve MUMPS' portability, extendability, flexibility, and completeness at the expense of stability. The changes make MUMPS more complete as a DBMS and programming language.

7.1 AST CATEGORY OF MUMPS

The AST definition enumerates the types of technologies by which the MUMPS programming language and environment may be categorized:

- Software Tool
- Life Cycle Support Environment
- Programming Support Environment
- Nonprocedural Language
- Data Base Management System
- Other Technology

7.1.1 SOFTWARE TOOL

A software tool is a computer program used in the development, testing, analysis, or maintenance of a program or its documentation. MUMPS is a programming language, and could be used to make software tools. However, MUMPS itself is not a software tool since it is a language, not a computer program. A MUMPS interpreter qualifies as a software tool. This study, however, documents the specifications of MUMPS with respect to AST, not implementations.

7.1.2 LIFE CYCLE SUPPORT ENVIRONMENT

A life cycle support environment is an integrated set of software tools and utilities that are designed to support a software product from the time it is conceived until it is no longer used. The MUMPS programming language standards do not incorporate life cycle support in their definition, nor do the programming language standards for many other languages such as Ada, C, Fortran, and Pascal. Commercial and proprietary life cycle support environments have been developed for MUMPS and other languages. A standard life cycle support environment targeted toward Ada has been proposed to DoD. DoD is expected to award a contract in 1993.

7.1.3 PROGRAMMING SUPPORT ENVIRONMENT

A programming support environment (PSE) is an integrated collection of software tools accessed through a single command language to provide programming support capabilities throughout the software life cycle. Through implementation-specific commands, MUMPS interpreters provide program loading, editing, and saving capabilities as in a PSE. Although these commands are not standardized, MUMPS contains a rudimentary PSE. DoD has developed a standard as well as a validation test suite for the interface to Ada PSEs; however, the interface standard is still separate from the language standard. Most other standard programming languages, such as C, Fortran, and Pascal, do not incorporate PSEs in their definitions. Commercial and proprietary PSEs have been developed for MUMPS as well as other languages, but none have been standardized.

7.1.4 NONPROCEDURAL LANGUAGE

A nonprocedural language is one in which the user states what is to be achieved, without stating specific instructions that the computer must execute in a given sequence. Most programming languages, such as Ada and MUMPS, are procedural since the programmer explicitly sequences the commands that the computer must execute. Some data base languages, such as SQL, are nonprocedural, since the user specifies only the desired result and the data base manager automatically determines how to execute the command.

MUMPS has aspects of both procedural languages and nonprocedural languages. MUMPS uses routines to structure and sequence the instruction commands that the computer will execute. The MUMPS programming language is clearly a procedural language. However, the intrinsic MUMPS data base management commands are nonprocedural. For example, when the user directs MUMPS to store data, MUMPS automatically allocates sufficient storage space, assigns the data values, and updates indices to the new data. The MUMPS nonprocedural data base management commands typically are embedded within the procedural programming language for sequencing and structure. This is similar to embedding SQL statements within another programming language.

7.1.5 MODERN DATA BASE MANAGEMENT SYSTEM

MUMPS directly supports persistent data which is shared among users of the MUMPS system. MUMPS currently provides a simple record locking mechanism; full transaction processing has been proposed for the 1993 ANS. MUMPS can be evaluated as a data base management system.

The added data manipulation functions proposed for the 1993 ANS increase MUMPS' power as a DBMS. The enhanced \$Order functions, data base merge function, and capability to directly update the data base greatly improve MUMPS' capability to manipulate data. The SQL binding allows MUMPS to support industry and open systems standards.

7.1.6 OTHER TECHNOLOGY

The "other technology" clause in the definition of AST includes all technologies not covered among the explicitly enumerated technologies. MUMPS incorporates aspects of many of the above categories with a programming language. MUMPS can be considered a technology which integrates the AST technologies of a programming language, nonprocedural data management language, data base management system, and rudimentary programming support environment.

7.2 AST CAPABILITIES OF MUMPS

The changes to the MUMPS programming language and environment can be evaluated with respect to the following AST capabilities:

- Productivity
- Usability
- Maintainability
- Portability
- Flexibility
- Efficiency
- Extendibility
- Completeness
- Maturity
- Stability

Because MUMPS is a technology which integrates DBMS services with a programming language and a rudimentary PSE, a composite rating for each AST capability was established. To evaluate MUMPS against each of the above AST capabilities, the following qualitative scale was chosen:

- Excellent
- Good
- Average
- Fair
- Poor

A rating of "Excellent" means that MUMPS is clearly superior to the typical system commonly in use. Conversely, a rating of "Poor" means that MUMPS is clearly inferior to the typical system commonly in use. Exhibit 7-1 uses this scale to summarize the ratings of MUMPS against the AST criteria. In some cases MUMPS provides superior capabilities to typical systems; in other cases, MUMPS is less capable than typical technologies.

| EXHIBIT 7-1: MUMPS AST CRITERIA ASSESSMENTS | | | |
|---|-----------------------|-------------------------|------------|
| AST CRITERIA | 1984 ANS 1986 FIPS | 1990 ANS (1993 FIPS) | (1993 ANS) |
| Productivity | Average | Good | Good |
| Usability | Average | Average | Good |
| Maintainability | Fair | Average | Average |
| Portability | Good | Excellent | Excellent |
| Flexibility | Average | Average | Average |
| Efficiency | Average | Good | Good |
| Extendability | Fair | Fair | Average |
| Completeness | Fair | Average | Good |
| Maturity | Fair | Average | Average |
| Stability | Average | Fair | Fair |

7.2.1 PRODUCTIVITY

Productivity is the ease with which systems, programs, component functions, and procedures can be developed and written using MUMPS.

The productivity of MUMPS has been improved with the addition of block structuring in the changes to the MUMPS FIPS PUB. The addition of external routine calls and additional data retrieval functions will result in minor improvements in productivity.

7.2.2 USABILITY

Usability is the ease with which a user can learn the programming language, write correct programs in the language, and understand programs in the language.

The planned additions to the MUMPS FIPS PUB will improve the usability of MUMPS. The addition of block structuring improves a user's ability to read, write, and understand programs in the language. The bindings to industry and open systems standards, such as SQL and GKS, further improve a programmer's ability to write and understand programs in the language. The standardized bindings replace implementation-specific methods.

7.2.3 MAINTAINABILITY

Maintainability is the ease with which a program written in MUMPS can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment.

The addition of block structuring to the MUMPS FIPS PUB greatly improves the maintainability of MUMPS programs.

The planned changes to the MUMPS ANS should increase slightly the maintainability of MUMPS programs. The ability to access external routine libraries improves maintainability by reducing the amount of code required within the MUMPS application. The standardized error processing and handling will allow faults to be located more quickly. The improved parameter passing makes it easier to modify and understand the parameters passed to a routine.

7.2.4 PORTABILITY

Portability is the ease with which a computer program written in MUMPS can be transferred from one hardware environment, software environment, or vendor implementation to another. Traditionally, ANS MUMPS has had very good portability.

The proposed 1993 ANS will further increase portability by standardizing the method for calling external routines. Support for X-Windows, SQL, and GKS bindings will improve portability by allowing MUMPS programs to use system-independent libraries. The standardization of error processing and handling removes the vendor- and implementation-specific methods that were used previously. The standardized device handling, networking support, and distributed data support remove the need to use implementation-specific code for these functions.

7.2.5 FLEXIBILITY

Flexibility is the ease with which MUMPS programs can be modified for use in applications or environments other than those for which they were specifically designed.

The proposed 1993 ANS will increase flexibility slightly through support for X-Windows, SQL, and GKS bindings. The reduced dependence upon a specific system will allow MUMPS programs to be adapted more easily to systems with different hardware, operating systems, or vendor implementations.

7.2.6 EFFICIENCY

Efficiency is the degree to which MUMPS programs perform their functions with a minimal consumption of storage space and time. Traditionally, MUMPS has had excellent storage efficiency and good execution efficiency.

The planned changes to FIPS PUB 125 will improve the efficiency of MUMPS code. The extrinsic functions and external routines should result in faster execution of MUMPS code. The extrinsic functions can be implemented efficiently using the hardware support for advanced mathematics. The support for external libraries and routines which can be optimized for the underlying hardware should provide slightly improved performance.

7.2.7 EXTENDABILITY

Extendability is the ease with which the language's functional capabilities can be increased.

The planned changes to FIPS PUB 125 slightly increased the extendability of MUMPS code. The addition of extrinsic functions expands the number of functions that MUMPS can support.

The extendability of MUMPS also will be increased through the standardized external program calling method of the proposed 1993 ANS. MUMPS will be able to access external routines for statistical or mathematical functions.

7.2.8 COMPLETENESS

Completeness is the degree to which the standard provides definitions for key features that support programming services. Traditionally, MUMPS has had strong string and text handling but weaker capabilities for mathematical and statistical calculations. Bindings to MUMPS add graphics and networking.

The MUMPS FIPS PUB will become more complete with the addition of extrinsic functions. Extrinsic functions allow more efficient handling of complex mathematical functions within MUMPS, and provide a mechanism to further extend the language's functionality.

The planned 1993 ANS will further increase MUMPS' completeness as a DBMS and as a programming language. Full transaction processing support, data base manipulation, and standardized error processing and handling methods increase DBMS completeness. Additional string handling functions and pattern matching increase MUMPS' string and text handling completeness. Support for complex numbers, exponentiation, external routines, and external libraries increase MUMPS' completeness as a programming language. Two-dimensional graphics were added to MUMPS through the GKS binding, and OMI adds network support for distributed data bases.

7.2.9 MATURITY

The planned changes to FIPS PUB 125 and the MUMPS Language Standard reflect the increased maturity of the MUMPS programming language. MUMPS has undergone three ANS standardizations since its inception in 1967, and a fourth standardization is planned for 1993.

7.2.10 STABILITY

The planned changes to FIPS PUB 125 and the MUMPS Language Standard reveal that MUMPS is not stable in the near term, since some major changes are planned within the next two years. However, the anticipated changes will help make MUMPS more stable in the long term, since general support for external libraries and procedures was added. MUMPS will be able to integrate with industry and open systems environment standards.

7.3 CONCLUSIONS

Both the MUMPS FIPS PUB and the MUMPS Language Standard are undergoing revisions planned for 1993. These planned changes to FIPS PUB 125 and the MUMPS ANS primarily increase MUMPS' portability, extendability, flexibility, and completeness at the expense of its short-term stability. The addition of external library and routine support, standardized device handling, and interfaces to industry and open systems will result in improved portability, extendability, and flexibility. Standardized transaction processing, standardized error processing and handling, and additional data base functions will make MUMPS more complete as a DBMS. Better support for mathematical functions and additional string handling functions will make MUMPS more complete as a programming language.

APPENDIX A.1: ABBREVIATIONS

| <u>Acronym</u> | <u>Meaning</u> |
|----------------|---|
| ANS | American National Standard |
| ANSI | American National Standards Institute |
| AST | Advanced Software Technology |
| DBMS | Data Base Management System |
| DoD | Department of Defense |
| DoDD | Department of Defense Directive |
| FIPS | Federal Information Processing Standard |
| FIPS PUB | Federal Information Processing Standard Publication |
| GKS | Graphical Kernel Services |
| GUI | Graphical User Interface |
| HOL | Higher Order Language |
| IEC | International Electrotechnical Committee |
| IEEE | Institute of Electrical and Electronics Engineers |
| IHS | Indian Health Service |
| ISO | International Standards Organization |
| MDC | MUMPS Development Committee |
| MIL-STD | Military Standard |
| MUG | MUMPS Users Group |
| MUMPS | Massachusetts General Hospital Utility Multi-Programming System |

| | |
|-------|--|
| NIST | National Institute of Standards and Technology |
| OMI | Open MUMPS Interconnect |
| PSE | Programming Support Environment |
| RDBMS | Relational Database Management System |
| SQL | Structured Query Language |
| VA | Department of Veterans Affairs |

APPENDIX A.2: REFERENCES

A.2.1 MUMPS LANGUAGE STANDARDS AND SPECIFICATIONS

The Complete MUMPS: an Introduction and Reference Manual for the MUMPS Programming Language. John Lewkowicz, Prentice Hall. 1989.

ABCs of MUMPS. Dr. Richard F. Walters, Digital Press. 1989.

ANSI/MDC X11.1-1984 MUMPS Language Standard. Mumps Development Committee. 1984.

A.2.2 TECHNICAL REFERENCES

"Glossary of Software Engineering Terminology". IEEE Std 610.12-1990. *IEEE Software Engineering Standards Collection.* Spring 1991.

A.2.3 DOD POLICY REFERENCES

"Delegation of Authority and Clarifying Guidance on Waivers from the Use of the Ada Programming Language". Assistant Secretary of Defense: Command, Control, Communications, and Intelligence. 17 April 1992.

"Computer Programming Language Policy". DoD Directive 3405.1. 2 April 1987.

ANALYSIS OF MUMPS AS A HIGHER ORDER LANGUAGE

TABLE OF CONTENTS

| | Page |
|--|------|
| LIST OF EXHIBITS. | iii |
| EXECUTIVE SUMMARY | ES-1 |
| 1.0 INTRODUCTION | 1-1 |
| 1.1 Scope Considered | 1-1 |
| 1.2 Methodology | 1-4 |
| 1.3 Data Sources | 1-7 |
| 1.4 Document Organization | 1-7 |
| 2.0 EVALUATION CRITERIA | 2-1 |
| 2.1 NIST APP and DoD TRM Criteria | 2-1 |
| 2.2 NBS HOL Criteria | 2-3 |
| 3.0 LANGUAGES CONSIDERED | 3-1 |
| 3.1 Level of Consensus | 3-1 |
| 3.2 Product Availability | 3-3 |
| 3.3 Completeness | 3-3 |
| 3.4 Maturity | 3-4 |
| 3.5 Stability | 3-5 |
| 3.6 De Facto Usage | 3-6 |
| 3.7 Problems and Limitations | 3-7 |
| 4.0 LANGUAGE CHARACTERISTICS | 4-1 |
| 4.1 Syntactic Style | 4-1 |
| 4.2 Semantic Structure | 4-2 |
| 4.3 Data Types and Manipulation | 4-6 |
| 4.4 Interfacing | 4-12 |
| 4.5 Extendibility | 4-13 |
| 4.6 Summary of Technical Characteristics | 4-16 |
| 5.0 APPLICATION REQUIREMENTS | 5-1 |
| 5.1 Functional Support | 5-1 |

TABLE OF CONTENTS

(Continued)

| | Page |
|-------------------------------------|------|
| 5.2 Size and Complexity | 5-1 |
| 5.3 Typical Life Cycle. | 5-4 |
| 5.4 Programming Expertise | 5-4 |
| 5.5 Many Developers | 5-5 |
| 5.6 End-User Interaction | 5-5 |
| 5.7 Reliability | 5-5 |
| 5.8 Portability | 5-6 |
| 5.9 Performance | 5-6 |
| 6.0 LANGUAGE SUITABILITY | 6-1 |
| 6.1 Functional Support | 6-2 |
| 6.2 Size and Complexity | 6-2 |
| 6.3 Typical Life Cycle | 6-3 |
| 6.4 Programming Expertise | 6-3 |
| 6.5 Many Programmers | 6-4 |
| 6.6 End-User Interaction | 6-4 |
| 6.7 Reliability | 6-4 |
| 6.8 Portability | 6-5 |
| 6.9 Performance | 6-5 |
| 7.0 CONCLUSION | 7-1 |
| 7.1 MUMPS | 7-1 |
| 7.2 Ada | 7-2 |
| 7.3 C | 7-2 |
| 7.4 COBOL | 7-3 |
| 7.5 FORTRAN | 7-3 |
| 7.6 Pascal | 7-4 |
| 7.7 Observations | 7-4 |

APPENDIXES

- A - References
- B - Acronym Definitions

LIST OF EXHIBITS

| Number | | Page |
|--------|---|------|
| 3-1 | NIST APP HOL Evaluation | 3-2 |
| 3-2 | Standardization. | 3-2 |
| 3-3 | Origins of Selected Programming Languages. | 3-5 |
| 4-1 | Syntactic Style | 4-1 |
| 4-2 | Execution Control | 4-3 |
| 4-3 | Data Control | 4-5 |
| 4-4 | Character Data and Operations | 4-7 |
| 4-5 | Numeric Data and Operations. | 4-8 |
| 4-6 | Logical Data and Operations. | 4-8 |
| 4-7 | Data Control Features. | 4-9 |
| 4-8 | Array Features. | 4-10 |
| 4-9 | Record Features. | 4-11 |
| 4-10 | File Input/Output. | 4-13 |
| 4-11 | Library Features | 4-14 |
| 4-12 | Language Feature Summary | 4-16 |
| 5-1 | Application Requirements and Evaluation Criteria. | 5-2 |
| 5-2 | Application Requirements and Language Features. | 5-3 |
| 6-1 | Language Vs. HOL Requirements | 6-1 |

EXECUTIVE SUMMARY

Department of Defense (DoD) policy, as stated in DoD Directive (DoDD) 3405.1, "Computer Programming Language Policy," is to construct software using the following tools, in descending order of preference:

- Off-the-shelf application packages and advanced software technology
- Ada-based software and tools
- Approved standard Higher Order Languages (HOLs)

Much of the software developed for the Military Health Services System has been created using the Massachusetts General Hospital Utility Multi-Programming System (MUMPS). This paper evaluates MUMPS against the criteria defined for an HOL. MUMPS is evaluated against Ada and other commonly used or DoD-approved HOLs (C, COBOL, FORTRAN, and Pascal).

The National Institute of Standards and Technology (NIST) Application Portability Profile (APP) and the DoD Technical Reference Model (TRM) specify seven criteria for evaluating HOL standards:

- Level of consensus
- Product availability
- Completeness
- Maturity
- Stability
- De facto usage
- Problems and limitations

In addition to being evaluated based on these criteria, MUMPS was compared to the other languages based on six criteria found in the National Bureau of Standards (now NIST) Special Publication on the *Selection and Use of General-Purpose Programming Languages*:

- Syntactic style
- Semantic structure
- Data types and manipulation
- Interfacing
- Extendibility
- Development and maintenance characteristics

Each of the languages evaluated was created for a specific purpose:

- MUMPS to handle unstructured text data
- Ada to create highly reliable embedded systems
- C for systems programming

- COBOL for business programming
- FORTRAN to perform scientific calculations
- Pascal for education

In addition to the general criteria for HOLs cited above, each language is evaluated based on suitability for its intended applications.

Overall, MUMPS appears to meet the HOL criteria as do three of the DoD-approved languages (COBOL, FORTRAN, and Pascal). The newer languages, C and Ada, appear generally to meet the criteria better than MUMPS. MUMPS appears to satisfy the criteria especially well when it is used for small to moderate-sized applications that deal primarily with shared unstructured or textual data, such as those often found in the medical functional area.

1.0 INTRODUCTION

The Office of Technical Integration has requested a study of the Massachusetts General Hospital Utility Multi-Programming System. This report focuses on one aspect of the study: how MUMPS rates against the Higher Order Language criteria and documentation required for Department of Defense approval.

For this study, MUMPS was considered with respect to DoD HOL criteria and documentation to determine the eligibility of MUMPS for inclusion on the list of DoD-Approved Higher Order Programming Languages. For comparative purposes, other standardized DoD-approved HOLs also were considered with respect to their meeting DoD HOL criteria.

The methodology chosen was to qualitatively evaluate each of these languages against a fixed set of evaluation criteria. Candidate evaluation criteria were contained in documents produced by the National Institute of Standards and Technology DoD, and the National Bureau of Standards (NBS, now NIST). Specific sources were as follows:

- NIST - *Application Portability Profile*
- DoD - *Technical Reference Model*
- NBS - *Selection and Use of General-Purpose Programming Languages*

The information in this report was gathered through personal interviews, telephone conversations, and a literature search. Staff at NIST and Robert Morris College were interviewed about MUMPS and other HOLs. Discussions with MUMPS Users Group (MUG) contacts and MUMPS Development Committee (MDC) members provided information about MUMPS. A literature review provided information about DoD HOL criteria and other methods for comparing HOLs.

1.1 Scope Considered

DoD HOL selection is governed by DoD internal policy and overall Federal policy and guidelines. Federal policy toward HOL selection is covered by the guidelines and standards detailed in the National Institute of Standards and Technology Application Portability Profile. DoD has elected to develop its own set of policies that go beyond the NIST APP, specifying alternative or additional HOLs to be conditionally acceptable. These alternative and additional HOLs, and the conditions for their use, are given in DoD Directive 3405.1.

1.1.1 FEDERAL GUIDELINES

NIST has defined a set of standards to assist Federal agencies in selecting and using Open Systems Environment (OSE) specifications. An OSE provides a nonproprietary standards-based framework for interoperable, portable, and scalable applications. An OSE allows the Government to integrate its applications without dependence on a particular vendor. The NIST APP specifies the set of OSE specifications that NIST has chosen for the Federal Government. This set of interfaces, services, protocols, and data formats provides guidance to Federal users by defining a suite of standards from which Federal users can select.

The NIST APP states the following:

The APP is *not* a standard and is not designed to cover every case. In some instances, the selection of one specification recommended in the APP will obviate the need for other specifications that are also recommended, but for somewhat different user requirements. In areas where the APP does not meet all of a user's requirements, users must augment the recommended specifications to ensure that proposed systems meet their requirements.

The NIST APP recommends five standardized HOLs for Federal use:

- Ada
- C
- COBOL
- FORTRAN
- Pascal

While NIST has recommended this list of standardized HOLs, Federal users may develop their own policies toward HOLs.

1.1.2 DoD TECHNICAL POLICY

In a memorandum dated 12 February 1992, the Director of Defense Information (DDI) specified DoD policy for information systems. This memorandum, titled "Open Systems Implementation and the Technical Reference Model," required all DoD components to apply the Technical Reference Model. The DoD TRM specifies standards to increase commonality and interoperability of information systems within DoD. The DoD TRM is based on NIST standardization activities and uses the NIST APP as a baseline, since DoD is required to adhere to the standards promulgated by NIST.

Through application of the DoD TRM in an OSE, DoD expects to accomplish the following:

- Improve user efficiency
- Improve development efficiency
- Improve portability and scalability
- Improve interoperability
- Promote vendor independence
- Reduce life cycle costs
- Improve security

The DoD TRM meets DoD-specific needs that were not addressed in the NIST APP. The DoD TRM uses the same criteria as the APP, but with greater emphasis on maturity, stability, completeness, and availability, since these characteristics are more important to DoD. The DoD TRM specifies Ada as the only DoD HOL.

1.1.3 DoD HOL POLICY

In a memorandum dated 17 April 1992, the Assistant Secretary of Defense (Command, Control, Communications, and Intelligence (C3I)) clarified the DoD policy on HOLs. This memorandum, titled "Delegation of Authority and Clarifying Guidance on Waivers from the Use of the Ada Programming Language," required all new development or modification of DoD software to comply with DoD Directive 3405.1.

DoDD 3405.1, "Computer Programming Language Policy," states that DoD policy is to prefer, based on life cycle costs and impact, the use of the following:

- Off-the-shelf application packages and advanced software technology
- Ada-based software and tools
- Approved standard Higher Order Languages

DoDD 3405.1 further states:

When Ada is not used, only the other standard higher order programming languages shown in enclosure 3 shall be used to meet custom-developed procedural language programming requirements. The use of specific HOLs shall be based on capabilities of the language to meet system requirements. Guidance in selecting the appropriate HOL to use is provided in NBS Special Publication 500-117.

Enclosure 3 of DoD Directive 3405.1 lists the following HOLs as acceptable for use by DoD:

- Ada
- C/ATLAS
- COBOL
- CMS-2M
- CMS-2Y
- FORTRAN
- JOVIAL
- Minimal BASIC
- Pascal
- SPL/1

Since these languages have been designated as acceptable for DoD use, it can be inferred that they were considered with respect to DoD HCL criteria and documentation and found to be eligible based on these criteria.

1.2 Methodology

For this study, MUMPS was considered with respect to DoD HOL criteria and documentation to determine the eligibility of MUMPS for inclusion on the list of DoD-Approved Higher Order Programming Languages. As stated in DoDD 3405.1, NBS Special Publication (SP) 500-117, *Selection and Use of General-Purpose Programming Languages*, is to be used in selecting an appropriate HOL based on that language's capabilities.

NBS SP 500-117 defines broad areas in which HOLs can be compared. Sets of technical language features are further defined within these broad areas against which each language can be compared with the other languages. After determining the technical strengths and weaknesses of each of the languages, these strengths and weaknesses are mapped to functional application requirements. These application requirements ultimately determine the suitability of a specific language to a particular application.

The NIST APP and DoD TRM provide other criteria for evaluating HOL standards. These criteria allow standards to be evaluated for usage by the Government.

MUMPS was compared to a subset of the DoD-approved HOLs and NIST-recommended HOLs. Thus, the similarities and differences among HOLs could be highlighted, providing a sharper characterization of each language. For this study, the following criteria were established to determine whether a language should be included in the subset for comparative purposes:

- The language is an ANSI-approved standard.
- Current information about the HOL is readily available.

The first step was to determine whether an unambiguous definition of the language existed. If ANSI had approved the language standard, then the language was presumed to be well defined for usage within the United States.

The second step was to determine whether current information about the language was readily available. If the language information was not current, then the language may have evolved and factual errors from using obsolete information would result.

Based on the above criteria, the following languages were considered in addition to MUMPS:

- Ada
- C
- COBOL
- FORTRAN
- Pascal

The methodology chosen was to evaluate each of these languages against a fixed set of criteria using a qualitative scale. For this study, language criteria were drawn from the following sources:

- NIST APP - *Application Portability Profile*
- DoD TRM - *Technical Reference Model*
- NBS - *Selection and Use of General-Purpose Programming Languages*

1.2.1 MUMPS

MUMPS was developed by Massachusetts General Hospital to manage clinical recordkeeping. MUMPS design goals were as follows:

- To share text data in an interactive environment
- To store sparse variable-length textual data efficiently
- To provide good performance on minicomputers
- To integrate a data base with the programming language

MUMPS is best known for its ability to handle unstructured textual information, its data sharing capabilities among MUMPS programs, and the interpretive nature of the language.

1.2.2 ADA

Ada was developed to meet DoD needs for large, embedded real-time weapons systems. The design objectives were as follows:

- To help maintain large applications
- To support software reuse
- To incorporate existing software engineering concepts
- To support highly robust applications

1.2.3 C

C was developed by Dennis Ritchie of Bell Labs in 1972 for portable and efficient operating systems and systems support. The design objectives were as follows:

- To improve integer and floating-point handling
- To provide efficient implementation

1.2.4 COBOL

COBOL was developed by a DoD-sponsored committee in 1960 to support business data processing. The design objectives were as follows:

- To use English-like syntax
- To provide ease of use
- To be independent of implementation details

1.2.5 FORTRAN

FORTRAN was developed by IBM in 1954 to develop scientific calculation applications. FORTRAN is a high-level programming language characterized by weak data typing and strong matrix array support. The design objectives were as follows:

- To provide efficiency roughly equivalent to hand-coded machine language
- To provide programming ease equivalent to interpreted methods

1.2.6 PASCAL

Pascal was developed by Niklaus Wirth for education purposes in 1971, incorporating structured programming and strong typing concepts. The design objectives were as follows:

- To demonstrate structured control concepts
- To demonstrate abstract data types and clear data scoping
- To provide an elegant syntax and efficient compilation

1.3 Data Sources

The information in this report was gathered through personal interviews, telephone conversations, and a literature search. NIST staff members were interviewed to determine whether MUMPS had already been evaluated. NIST concluded that MUMPS currently did not have enough usage to warrant an evaluation. Contacts provided by MUG and MDC members were consulted to gather information about the MUMPS programming language and environment. Contacts at Robert Morris College were interviewed to gather additional information about the MUMPS programming language and environment from an academic viewpoint. The MUMPS Language Standard (ANSI/MDC X11.1-1990) was consulted for details about the MUMPS language and portability standards. Other studies related to MUMPS were examined for relevance to this analysis.

Ada information was compiled primarily from written sources. In addition to the Ada language standard (DOD-STD-1813, 1983), numerous texts and accounts by Ada designers were consulted. Newsletters and documents from the Ada Information Center provided valuable insight into the current status of Ada bindings and the proposed revised language standard, Ada 9x.

In addition, a review of literature on programming language history, design, and theory was performed to gather information about C, COBOL, FORTRAN, and Pascal. A full listing of sources is presented in appendix A.

1.4 Document Organization

The remainder of this document is organized as follows. Chapter 2.0 defines the criteria used to evaluate programming languages. These criteria will form the basis for the analysis of MUMPS and other languages in the later chapters.

Chapter 3.0 describes how MUMPS and the other languages rate with respect to the criteria defined by NIST. Chapter 4.0 compares and contrasts the technical characteristics of each of the languages. Chapter 5.0 introduces functional application requirements. Chapter 6.0 examines

the relationship between technical language features and functional application requirements, while chapter 7.0 summarizes the previous chapters by examining each of the languages.

Appendix A lists references; appendix B defines the acronyms used in this report.

2.0 EVALUATION CRITERIA

The methodology chosen was to qualitatively evaluate each of these languages against a fixed set of criteria. For this study, the evaluation criteria were drawn from the following sources:

- NIST - *Application Portability Profile*
- DoD - *Technical Reference Model*
- NBS - *Selection and Use of General-Purpose Programming Languages*

The DoD TRM and the NIST APP use the same criteria to evaluate standards for HOLs and other components. The NBS criteria are designed specifically to evaluate HOLs on the basis of their technical and functional merits.

2.1 NIST APP and DoD TRM Criteria

Both the NIST APP and DoD TRM identify the following criteria for evaluating HOL standards:

- Level of consensus
- Product availability
- Completeness
- Maturity
- Stability
- De facto usage
- Problems and limitations

Each of these evaluation criteria is discussed below.

2.1.1 LEVEL OF CONSENSUS

This criterion measures how widely accepted the language standard is. Proprietary specifications, or those with very limited or specialized usage, have a low level of consensus. Conversely, national and international standards have a high level of consensus.

2.1.2 PRODUCT AVAILABILITY

Product availability is the degree to which compilers or interpreters that conform to the language standard are widely available. Product availability is also influenced by the number of programming support tools that support the language. If only a few proprietary products are

available, then product availability is low. However, if numerous products that conform to the language standard are available from various vendors on many platforms, then product availability is high.

2.1.3 COMPLETENESS

Completeness is the degree to which a language standard defines and covers key features necessary to support modern programming styles, such as structured programming; data abstraction, such as user-defined data types; and functional abstraction, such as high-level functions.

2.1.4 MATURITY

Maturity is the degree to which a specification and its underlying technologies and concepts are well understood. If the specification is defined by a formal mathematical model or the technology may have been used for many years, then it has high maturity. Conversely, if a language is relatively new or is not well defined, then it has low maturity. A lower rating would also apply if the language represents an emerging technology or is in an early phase of its life cycle and has not been refined through practical application.

2.1.5 STABILITY

Stability is the likelihood that the specification will not have many or significant changes within the next 2 years. If the language specification is expected to have many significant changes within a short period of time or changes that introduce notable incompatibilities with features that are still in use, then the language has low stability. If the language specification is not expected to change, then the language has high stability.

2.1.6 DE FACTO USAGE

This criterion measures the extent to which users have used the language, currently use the language, or are familiar with the language. If only a few groups of people use the language, then it has low de facto usage. If the language is widely known and used by a majority of people, then it has high de facto usage.

2.1.7 PROBLEMS AND LIMITATIONS

This criterion measures the extent to which the language can be used without restrictions. If the language is limited in its capabilities, is restricted in use, or has problems that are exceptionally hard to overcome, then it would receive a low rating. If the language is fully effective, then it would receive a higher rating.

2.2 NBS HOL Criteria

Based on the NBS report on the selection criteria for languages, the following technical characteristics of HOLs are discussed below:

- Syntactic style
- Semantic structure
- Data types and manipulation
- Interfacing
- Extendibility

2.2.1 SYNTACTIC STYLE

Syntactic style refers to the features of a language that bear on its readability and appearance but have no direct bearing on the control or data structures. It includes line and page layout, size and character of identifiers and labels, and requirements for explicit declarations.

2.2.2 SEMANTIC STRUCTURE

Semantic structure refers to features of a language that describe the structure of the algorithms and the logical appearance of data. Clarity and correctness are enhanced when a language's semantic structure is robust; while work-arounds are often possible, they are potential sources of error. Semantic structure consists of execution control and data control.

All modern programming languages allow the programmer to decompose the execution sequence into logical groups in order to more clearly express the underlying design. Major methods include control structures, blocks, modules (subroutines or functions), exception handling, and concurrency.

Data control deals with the scope, lifetime, and other logical properties of data. These capabilities allow the programmer to establish the logical appearance of data within a program. Data scope may be local to a module, shared among a group of modules, or global to an entire program. Data lifetime may be either static (available throughout the execution of the program) or dynamic (created and destroyed in the course of program execution).

2.2.3 DATA TYPES AND MANIPULATION

Data types may be considered as classes of objects that have a domain of allowed values and associated operations. This discussion will focus first on the types provided directly by each language. It then will consider the capabilities each language provides to allow the programmer to define types and operations. Finally, it will consider each language's facilities for constructing composite data types.

Type checking is a language feature that allows only restricted combinations of data types and operations. In its strictest form, it limits operations to operands of the same type. This ensures desired results and assists in detecting programmer errors.

Type coercion, on the other hand, allows a wide variety of interaction among data types and operations. Rules govern the conversion of operands in mixed-type expressions. Coercion may yield power and flexibility for the programmer but incurs increased risk of error.

2.2.4 INTERFACING

As used here, interfacing describes the capabilities a language offers for communicating with external processes or persons. Interactive support describes a language's facilities for communicating with a person at a computer terminal or similar device; file input/output (I/O) describes access to organized collections of data that exist apart from the program, including those originating in other programs or computer systems.

2.2.5 EXTENDIBILITY

Extendibility is the capability of the language to support functions and other capabilities extrinsic to the language. The language's scope can be extended by the programmer and by libraries or bindings. The programmer is responsible for much of the extendibility of the language through the abstraction capabilities. Libraries and bindings provide the principal mechanisms for extending the capabilities of a programming language into specialized areas. In fact, it can be argued that the availability of standardized extensions is a major factor in a language's economy, power, and flexibility.

3.0 LANGUAGES CONSIDERED

This section discusses the languages considered in this report. For comparison with MUMPS, this report concentrates on languages that have been standardized and approved for DoD usage. The following HOLs met the criteria defined in Chapter 1.0 for comparison with MUMPS:

- Ada
- C
- COBOL
- FORTRAN
- Pascal

For each language, the following characteristics from the NIST APP are discussed below:

- Level of consensus
- Product availability
- Completeness
- Maturity
- Stability
- De facto usage
- Problems and limitations

The evaluations of each language are summarized in exhibit 3-1. Exhibit 3-1 is based on information drawn from the NIST APP but for purposes of this report includes information on MUMPS.

3.1 Level of Consensus

Consensus is considered high when a standard for a language exists. As seen in exhibit 3-2, most of the HOLs are not only national standards, but also Federal and international standards. Within the United States, the national standards body is the American National Standards Institute (ANSI). The National Institute of Standards and Technology approves Federal standards by issuing a Federal Information Processing Standard (FIPS) Publication (FIPS PUB). The International Standards Organization/International Electrotechnical Commission (ISO/IEC) approves standards for international usage.

All of the languages considered in this report have been approved by ANSI and have an associated FIPS PUB. All of the languages, except for C, have been approved by the ISO as international standards. The ANSI standard for C is relatively recent and it is being considered by the ISO as an international standard.

| EXHIBIT 3-1: NIST APP HOL EVALUATION | | | | | | |
|--------------------------------------|-------|-----|---|-------|----------|--------|
| EVALUATION CRITERIA | MUMPS | Ada | C | COBOL | FORTTRAN | Pascal |
| Level of Consensus | ● | ● | ● | ● | ● | ● |
| Product Availability | — | ● | ● | ● | ● | ○ |
| Completeness | ○ | ● | ● | ● | ● | — |
| Maturity | ● | ● | ● | ● | ● | ● |
| Stability | ○ | ○ | ● | ● | ● | ● |
| De Facto Usage | — | ○ | ● | ● | ● | — |
| Problems and Limitations | — | ○ | ● | ● | ● | — |
| Key: ● High ○ Average — Low | | | | | | |

| EXHIBIT 3-2: STANDARDIZATION | | | | | | |
|---|-----------|----------------|---------|------------------------|----------|--------|
| STANDARD | MUMPS | Ada | C | COBOL | FORTTRAN | Pascal |
| FIPS PUB | 125 | 119 | 160 | 021-3 | 0691 | 109 |
| Date | 1990 | 1985 | 1991 | 1989 | 1989 | 1985 |
| ANSI | MDC X11.1 | MIL-STD- | X3J11 | X3.23 | X3.9 | X3.97 |
| Date | 1990 | 1815A: 1983 | 1989 | 1985 X3.23A 1989 | 1978 | 1983 |
| ISO | 11756 | 8652 | pending | 1989 | 1539 | 7185 |
| Date | 1991 | 1987 | | 1985 | 1980 | 1983* |
| * Includes and extends the ANSI/FIPS standard | | | | | | |

3.2 Product Availability

Product availability is high when a language is supported by many interpreters or compilers and related development tools.

In comparison to the other languages considered in this report, MUMPS is supported by a smaller number of products. MUMPS interpreters and systems are available from about 100 vendors. While this number is small compared to other languages, the existing products provide good coverage of major hardware platforms. Similarly, the development tools available for MUMPS are less numerous than for some more widely used languages.

Validated Ada compilers and environments are available from about 400 vendors. Many development tools are available, and additional tools will be developed by the DoD through its Integrated Computer-Aided Software Engineering (I-CASE) procurement. In addition to major hardware platforms, Ada compilers are available that target numerous processors used in embedded systems or control applications. Ada tools are available for all phases of the software life cycle, and include tools for monitoring and testing embedded systems.

As might be expected from their wide use in commercial and Government systems, C, COBOL, and FORTRAN compilers and related tools are widely available. Approximately 1,000 vendors offer C compilers and related development tools. Availability of compilers and tools for COBOL and FORTRAN is similarly high for all information system platforms. Tools for all phases of the life cycle are available from various vendors.

Pascal compilers and tools are widely available although not so numerous as those for C, COBOL, and FORTRAN. While products are available for all hardware platforms and life cycle phases, total availability falls between that of products for MUMPS and Ada.

3.3 Completeness

While all the languages considered in this report are reasonably complete for use as general-purpose programming languages, they differ in their particular strengths and weaknesses.

MUMPS is most complete in the area of string handling and data manipulation. It contains an integrated data base management system and has limited support for concurrency. MUMPS lacks advanced mathematical and statistical capabilities. Its library support and graphical interface support are currently being standardized.

Ada is complete for use as a general-purpose language. It also supports embedded systems. Ada supports high-level functional abstraction, data type definition, and robust concurrency control.

Additional features, including improved support for object-oriented programming, are included in a proposed revision.

C is complete as a general-purpose programming language; it includes facilities for both high- and low-level programming. C supports low-level hardware control and bit manipulation in addition to high-level abstract functions and procedures. C also includes data structuring, library support, and memory management. It lacks standardized concurrency control.

COBOL is most complete for business operations; its strengths are in data manipulation and output formatting. COBOL lacks concurrency, low-level functions, and communications capabilities.

FORTRAN is complete as a general-purpose programming language with particular strength in numeric matrix calculations required for many scientific applications. It lacks support for advanced data types and concurrency facilities. FORTRAN is also weak in string handling.

Pascal is complete as a general-purpose high-level programming language. Pascal lacks concurrency control. Although the Pascal standard lacks separate compilability and advanced file input/output capabilities, these capabilities are frequently provided by vendor extensions.

3.4 Maturity

All of the languages are relatively mature, having been refined for at least 10 years each. In addition, all of the languages now incorporate modern software engineering principles to some extent. Newer languages such as Ada and Pascal incorporate these principles to a greater extent than do older languages such as COBOL or FORTRAN. Exhibit 3-3 summarizes the maturity of the languages evaluated.

MUMPS was originally developed in the late 1960s from predecessor languages such as JOSS. The MUMPS standard has been updated more frequently (in 1987 and 1990) than the other languages considered in this paper.

DoD sponsored the development of Ada based on extensive study and design competition that specified Pascal as a suitable predecessor. While the language standard has been unchanged since 1983, considerable effort has been placed on developing related standards for tools and for bindings to Portable Open Systems Interface Standard (POSIX) and other standards. A standard POSIX binding was adopted in 1992; additional standard bindings also have been adopted.

C was developed in the late 1970s from predecessor languages B and BCPL. The first true standard was adopted in 1989 after extensive study and discussion.

| EXHIBIT 3-3: ORIGINS OF SELECTED PROGRAMMING LANGUAGES | | | | |
|--|-----------|--|----------------------|---|
| Language | Year | Originator | Predecessors | Intended Purpose |
| FORTRAN | 1954-1957 | J. Backus (IBM) | (None) | Numeric computation |
| COBOL | 1959-1960 | Committee | (None) | Business data processing |
| MUMPS | 1967 | Mass. General Hospital | JOSS | Shared variable-length text data, multi-user interactive |
| Pascal | 1971 | N. Wirth (ETH Zurich) | ALGOL 60 | General and educational purposes. Supporting structured programming |
| C | 1974 | D. Ritchie (Bell Labs) | ALGOL 68 BCPL | Systems programming |
| Ada | 1979 | J. Ichbiah et al. (CII Honeywell Bull) | Pascal, SIMULA 67 | General-purpose, embedded applications, real time |

COBOL is among the oldest standard programming languages, and was established in the 1960s by DoD initiative. The COBOL standard was revised in 1974 and 1983.

FORTRAN, one of the first compiled languages, was developed in 1954 by IBM. FORTRAN was first standardized in the 1960s. Revisions of the FORTRAN standard were completed in 1978 and 1985.

Pascal was developed in the late 1970s, based on well-established structured programming concepts and predecessor languages. Pascal was standardized in 1983.

3.5 Stability

As defined by the NIST APP, stability is the likelihood that the standard will not have numerous or major changes within the next 2 years.

The MUMPS standard is being revised and is expected to be submitted to ANSI in 1993. This revision will contain support for external libraries and for bindings to external standards. It also will provide for access to data that is distributed among MUMPS systems. A further revision to the MUMPS standard is planned for 1996.

The Ada language standard is also being revised (as Ada 9X) and is expected to be submitted to ANSI and ISO in 1993. In addition to numerous clarifications and extensions, it will include improved support for object-oriented programming. Additional standardization will be provided by bindings to other standards.

The C standard is relatively recent, so few changes are currently anticipated.

The COBOL standard is undergoing revision to support improved communications interfaces and screen management. The timetable for submission has not yet been established.

The contents and schedule of revisions to the Pascal and FORTRAN standards have not yet been determined.

3.6 De Facto Usage

De facto usage is a measure of how widely known and used the HOL is in academic, commercial, and Government circles. De facto usage is also affected by the number of people who know or support the language.

MUMPS is not widely taught within academic circles, nor is it widely used within the United States outside of the health care industry. MUMPS is supported by a user community that is smaller than those using other languages.

DoD has mandated that Ada be used in all new DoD software development. DoD has also mandated Ada for all DoD major upgrades in which more than one-third of the source code is to be changed. Ada has limited use in the commercial sector. Ada is beginning to be more widely taught within academic circles and more widely used in non-DoD applications.

C is widely used in commercial software development and is supplanting FORTRAN for scientific and engineering applications. C usage is growing rapidly in academia. C is widely taught within universities.

COBOL is still widely used for commercial and Government business applications because of the large installed code base. Most installed Federal software is written in COBOL, comprising 60 to 80 percent of the installed code base.

FORTRAN is widely used world wide, especially for engineering and scientific applications. FORTRAN has been the de facto standard for scientific and engineering applications; however, C is beginning to replace FORTRAN as the new de facto standard for scientific and engineering applications.

Pascal is widely taught within universities. It is the de facto standard for introductory programming in academia. Pascal is not widely used in commercial or Government applications.

3.7 Problems and Limitations

Most HOLs have some well-known problems and/or limitations. Only the major problems and limitations are discussed below. The problems and limitations associated with a particular language are discussed in more detail in chapter 4.0.

MUMPS is somewhat specialized toward text data base applications. MUMPS lacks standardized external libraries or advanced mathematical functions. Data exchange with non-MUMPS software has not been standardized.

Ada's problems and limitations are still unknown.

During the standardization process, C's problems and limitations were corrected to the extent practical.

COBOL is specialized toward business and financial applications. COBOL is very limited with respect to communications functions.

FORTRAN is loosely typed, and some errors are hard to debug.

Standard Pascal lacks separate compilability, library support, and advanced file input/output capabilities.

4.0 LANGUAGE CHARACTERISTICS

This chapter analyzes the MUMPS programming language and selected DoD Higher Order Languages with respect to their technical characteristics.

The technical characteristics to be considered were discussed in section 2.2; they include syntactic style, semantic structure, data types and manipulation, interfacing, and extendibility. Information is drawn largely from the NBS SP 500-117, and has been expanded and updated for this report.

4.1 Syntactic Style

Syntactic style refers to the features of a language that bear on its readability and appearance, but that have no direct bearing on the control or data structures. Exhibit 4-1 displays these features for the languages considered in this report.

| EXHIBIT 4-1: SYNTACTIC STYLE | | | | | | |
|--|-------|----------|-------|---------|----------|----------|
| LANGUAGE FEATURE | MUMPS | Ada | C | COBOL | FORTTRAN | Pascal |
| Free Format | No | Yes | Yes | No | No | Yes |
| Labels | Name | Name | Name | Name | Number | Number |
| Identifier Size | 8* | Any size | 31 | 30 | 6 | Any size |
| Implicit Declarations | Yes | No | No | No | Yes | No |
| Verbal Style | Terse | Verbose | Terse | Verbose | Terse | Average |
| * Defined in MUMPS portability standard; language standard allows any size | | | | | | |

The syntactic style of MUMPS is similar to that of FORTRAN in that program input is line oriented: commands must be completed on one line, or continuation must be explicitly indicated. COBOL is also line oriented, but continuations are implicit. MUMPS commands within the scope of an IF or ELSE must be combined on a line, or a parameterless DO command must be used. The scope of a parameterless DO is indicated with a line prefix. In contrast, Ada, C, and Pascal feature program input that is independent of line and page layout; a single command can span multiple lines, or multiple commands can appear on a single line. In all languages, source reformatting programs may be used to increase readability.

While the MUMPS language standard places no restrictions on identifier length, the portability standard restricts them to eight characters. This restriction is similar to that of FORTRAN, which restricts identifiers to six characters. This restriction to short identifiers often leads to the use of rather cryptic abbreviations. In contrast, Ada, C, COBOL, and Pascal allow longer identifiers that more clearly indicate their meaning.

Like FORTRAN, MUMPS allows implicit declaration of variables. This can lead to programming errors when an identifier is misspelled. Ada, C, COBOL, and Pascal require explicit declaration of variables and can detect such errors in compilation.

4.2 Semantic Structure

Semantic structure refers to features of a language that describe the structure of the algorithms and the logical appearance of data. Semantic structure consists of execution control and data control.

4.2.1 EXECUTION CONTROL

All modern programming languages allow the programmer to decompose the execution sequence into logical groups in order to more clearly express the underlying design. Major methods include control structures, blocks, modules (subroutines or functions), exception handling, and concurrency. Each language's support for these methods is displayed in exhibit 4-2.

4.2.1.1 Control Structures

Most modern programming languages provide a set of control structures that include IF-THEN-ELSE, a CASE (multiple selection) construct, and looping constructs.

MUMPS differs from other languages because of its unusual implementation of IF-THEN-ELSE that creates two potential sources of error. First, the MUMPS IF command sets a system variable \$TEST based on the value of the given argument; this variable also can be set by other commands, notably LOCK. Second, unlike other languages, the MUMPS ELSE command is not syntactically linked to the IF construct. A MUMPS ELSE is an independent command, simply branching when the system variable \$TEST is false. While concatenating simple IF and ELSE statements will yield the expected results, the use of nested IF's, subroutines, or certain other MUMPS commands can cause logic errors due to implicit reuse of the \$TEST variable.

| EXHIBIT 4-2: EXECUTION CONTROL | | | | | | |
|--|----------|------|------|-------|----------|----------|
| LANGUAGE FEATURE | MUMPS | Ada | C | COBOL | FORTRAN | Pascal |
| Structured Programming | Partial | ● | ● | ● | Partial | ● |
| Blocks | Partial | ● | ● | ● | Partial | ● |
| User-Defined Subroutines | Internal | Both | — | Both | External | Internal |
| User-Defined Functions | — | Both | Both | — | Both* | Internal |
| Exception Handling | — | ● | I/O | I/O | I/O | I/O |
| Concurrency | Partial | ● | — | — | — | — |
| * No local data for internal functions | | | | | | |
| Key: ● Yes — No | | | | | | |

In Ada, C, and Pascal, a CASE construct supports decisions that depend on multiple possible values of a discriminator variable. COBOL and FORTRAN have this capability in a more restricted form. Such a construct is not available in MUMPS. To accomplish a multiple choice in MUMPS requires a chain of IF statements, forcing the programmer to guard against the potential problem noted in the preceding paragraph. Thus, MUMPS puts a burden on the programmer that most languages take care of automatically.

The MUMPS looping construct (the FOR statement) is similar to that provided by FORTRAN; it depends on a control variable (typically a counter) rather than an arbitrary condition. As in FORTRAN, one can execute a GOTO command to transfer control to a label elsewhere in the program. MUMPS further provides a QUIT command that will conditionally terminate a loop; this provides the capability to simulate the REPEAT...UNTIL or WHILE...DO looping constructs that are available in other languages.

It should be noted that MUMPS requires that the statements governed by an IF, ELSE, or FOR command must be concatenated on a single line. This encourages the use of the parameterless DO command that MUMPS provides for blocking.

4.2.1.2 Blocks

Blocks allow several statements to be appended and treated as a single construct. Ada, C, and Pascal provide this structure and, further, allow data to be defined locally only to the block; this local data can be useful for scratch variables. MUMPS, like COBOL, provides blocks but no local data. (FORTRAN handles blocking in a more limited manner: it allows multiple statements only within an IF-THEN-ELSE or looping construct.) Furthermore, MUMPS uses a line prefix "." to delimit the scope of a block. This prefix must be repeated when blocks are nested, which creates a potential for logic errors. Other languages typically define blocks with BEGIN..END delimiters; this feature is less prone to error.

4.2.1.3 Modules

Most languages provide for user-defined subroutines or functions that may be invoked from multiple locations to accomplish a particular purpose. Functions generally return a single value, while subroutines may return any number of values. External modules may be separately compiled; they typically communicate by using parameters passed from the calling routine. Internal modules are typically compiled integrally with the routine that invokes them; they usually have access to all data that is local to the invoking module.

Ada provides for both internal and external subroutines (procedures) and functions, as well as modules called tasks (described in section 4.2.1.5). C provides internal and external functions. COBOL, FORTRAN, and Pascal provide both functions and subroutines. MUMPS supports user-defined subroutines only (the DO statement with parameters); they may be internal or external.

4.2.1.4 Exception Handling

Exception handling is the capability of detecting and responding to an anomalous condition during program execution. The exceptions may include arithmetic faults (such as division by zero), detection of user intervention, or input/output conditions such as end-of-file. Ada has a generalized capability that syntactically associates a special section of code (an exception handler) with each section of code that must detect exceptions. C, COBOL, FORTRAN, and Pascal have a limited capability to detect and respond to input/output exceptions. Additional capabilities are vendor or implementation dependent, not part of the language standard.

MUMPS' current exception-handling capabilities are vendor or implementation dependent; standardization is proposed for the 1993 revision.

4.2.1.5 Concurrency

Concurrency (also called multi-threading) is the capability to define parallel tasks; that is, processes or actions that may be executed simultaneously or in any sequence. In addition, the capability to synchronize specified tasks (e.g., to force task A to wait for the completion of task B) may also be provided. Ada is the only language considered here that fully supports concurrency. COBOL provides a limited capability for communication between asynchronous processes, but the capability to generate or control them is not covered by the language standard. Concurrency may be achieved in C, FORTRAN, or Pascal by using nonstandard calls to operating system services.

MUMPS provides limited support for concurrency through its JOB command. This command allows a MUMPS application to generate a new process and pass initial parameters to it. While the processes may communicate through common variables, also called semaphores, no synchronization mechanisms are provided by the language standard.

4.2.2 DATA CONTROL

Data control deals with the scope and lifetime properties of data. These capabilities allow the programmer to establish the logical appearance of data within a program. Data scope may be local to a module, shared among a group of modules, or global to an entire program. Data lifetime may be either static (available throughout the execution of the program) or dynamic (created and destroyed in the course of program execution). Data control capabilities are portrayed in exhibit 4-3.

| EXHIBIT 4-3: DATA CONTROL | | | | | | |
|---------------------------|---------|------|------|--------|----------------|---------|
| LANGUAGE FEATURE | MUMPS | Ada | C | COBOL | FORTTRAN | Pascal |
| Scope of Data: | | | | | | |
| Local to Module | ● | ● | ● | ● | ● | ● |
| Global to Program | ● | ● | ● | ● | If so declared | ● |
| Data Lifetime | Dynamic | Both | Both | Static | Static | Dynamic |
| Key: ● Yes — No | | | | | | |

4.2.2.1 Scope of Data

Ada, C, and Pascal support data that is local to specific modules and data that is global to an entire program. FORTRAN and COBOL provide this capability in more limited ways. In MUMPS, however, most data is contained in arrays that are available to the entire program. In fact, MUMPS calls such arrays "globals."

The ability to restrict the scope of data is useful in reducing potential errors. For example, where scope is restricted to a single module, a misspelled variable name will create a malfunction that is limited to that module. On the other hand, an error in referring to a global variable may affect the operation of every module that uses that variable.

4.2.2.2 Data Lifetime

Ada, C, and Pascal support dynamic allocation of data during program execution, typically through the use of pointers to a stack or heap. (Ada and C support static allocation as well.) Dynamic allocation is useful in allowing reuse of storage in the course of program execution. COBOL and FORTRAN, on the other hand, support only static data allocation; reuse of storage is more problematic with these languages.

MUMPS data is dynamic; variables and arrays are created and allocated when first referenced. They may be released through the use of the KILL command. MUMPS, like C and Pascal, requires the programmer to explicitly deallocate storage.

4.3 Data Types and Manipulation

Data types may be considered as classes of objects that have a domain of allowed values and associated operations. This discussion will focus first on the types provided directly by each language. It then will consider the capabilities of each language to define new types and operations. Finally, it will consider each language's facilities for constructing composite data types.

4.3.1 LANGUAGE-PROVIDED ATOMIC DATA TYPES

Atomic data (sometimes called scalars) is the smallest logical unit a language represents. MUMPS is distinctive among the languages considered here in that it supports only one atomic data type, the character string. It provides limited support for numeric data (integers or real numbers) in computations, but all data is stored as character strings and coerced to numeric format when needed. (A complex numeric type is included in the proposed 1993 language standard.) In similar fashion, a logical (true/false) data type is supported through coercion from stored character strings. Each language's support for character data is portrayed in exhibit 4-4.

| EXHIBIT 4-4: CHARACTER DATA AND OPERATIONS | | | | | | |
|---|----------|-----------------------|------------|----------|------------|--------|
| LANGUAGE FEATURE | MUMPS | Ada | C | COBOL | FORTTRAN | Pascal |
| Aggregate Character Data | Array | Array | Array | String | String | Array |
| Fixed/Variable Length | Variable | Fixed | Variable* | Fixed | Fixed | Fixed |
| Concatenation | ● | ● | ● | ● | ● | — |
| Substring | | | | | | |
| By position: | ● | ● | — | ● | ● | — |
| By contents: | ● | — | — | ● | — | — |
| String Search | ● | — | — | ● | ● | — |
| Test Upper/Lower Case, Numeric | ● | — | ●! | ● | — | — |
| Translate | ● | — | — | ● | — | — |
| Conversion Mechanism | Coercion | Functions, Pseudo-I/O | Pseudo-I/O | Coercion | Pseudo-I/O | None |
| * Within fixed-length array ! Individual characters only Key: ● Yes — No | | | | | | |

All other languages considered here provide a variety of types in addition to character strings. All provide a selection of numeric types, usually including integers and real numbers at both default and extended precision; some support complex numbers or fixed-length decimals. (See exhibit 4-5.)

In addition to numeric types, Ada, COBOL, FORTRAN, and Pascal support a logical (true/false) type; C implements a similar feature using integer values. Support for logical types is shown in exhibit 4-6.

| EXHIBIT 4-5: NUMERIC DATA AND OPERATIONS | | | | | | |
|--|-------|--------|----------|---------|----------|--------|
| LANGUAGE FEATURE | MUMPS | Ada | C | COBOL | FORTTRAN | Pascal |
| Long/Short Floating-Point | — | Both | Both | ● | Both | Short |
| Binary/Decimal Fixed-Point | — | Binary | — | Decimal | — | — |
| Integer | — | ● | ● | ● | ● | ● |
| Complex | — | — | — | — | ● | — |
| Add, Subtract, Multiply, Divide | ● | ● | ● | ● | ● | ● |
| Exponentiation | — | ●* | Function | ● | ● | — |
| Numeric Functions | None | Few | Many | None | Many | Some |
| * To integer power only | | | | | | |
| Key: ● Yes — No | | | | | | |

| EXHIBIT 4-6: LOGICAL DATA AND OPERATIONS | | | | | | |
|--|--------|---------|---------|------------------|----------|---------|
| LANGUAGE FEATURE | MUMPS | Ada | C | COBOL | FORTTRAN | Pascal |
| Logical Data Implementation | String | Logical | Numeric | Named Conditions | Logical | Logical |
| Logical Operations: | | | | | | |
| And, or, not | — | ● | ● | ● | ● | ● |
| Exclusive-or | — | ● | — | — | ● | — |
| Bit Data and Manipulation | — | — | ● | — | — | — |
| Key: ● Yes — No | | | | | | |

4.3.2 ABSTRACT AND USER-DEFINED ATOMIC DATA TYPES

Ada, C, and Pascal support the creation of abstract data types (ADTs) and user-defined types (UDTs). ADTs provide the capability of describing variables in terms of both their meaning and their domain of valid values. For example, defining a variable of type "velocity" confers more meaning than simply declaring it a real number of certain precision. Further, if a change is needed in the underlying domain, it can be made once and will propagate throughout the program. Ada, C, and Pascal provide for ADTs. Ada and Pascal automatically provide type checking, assignment, and comparison operations. Each language's support for ADTs and UDTs is shown in exhibit 4-7.

| EXHIBIT 4-7: DATA CONTROL FEATURES | | | | | | |
|------------------------------------|-------|-----|---|-------|----------|---------|
| LANGUAGE FEATURE | MUMPS | Ada | C | COBOL | FORTTRAN | Pascal |
| Data Abstraction: | | | | | | |
| User-Named Types | — | ● | ● | — | — | ● |
| Type-Checking | N/A | ● | — | N/A | N/A | ● |
| User-Defined Operations | N/A | ● | — | N/A | N/A | Partial |
| Key: ● Yes — No | | | | | | |

Ada provides a further capability: it allows a programmer to define new operations as well as new data types. The combination of ADTs and new operations yields UDTs. UDTs can provide powerful support for specialized problem domains.

MUMPS, in comparison, does not directly support ADTs or UDTs. While its character strings can be used for composite data (e.g., dates), the language provides no type checking or other support. MUMPS requires the programmer to explicitly specify any type checking in the program code, while other languages perform this function automatically. MUMPS gives programmers a flexibility that can be useful, but imposes a burden of responsibility for potential errors.

4.3.3 COMPOSITE DATA TYPES

Most of the languages considered here provide capabilities to build up aggregates or structures from smaller data units; these are called composite data. The principal composites are arrays, records, and sets.

Arrays are ordered collections of data of similar type. Each element can be addressed individually according to an identifying scalar value. Depending on the language, arrays may be nested or combined to represent two, three, or more dimensions. Exhibit 4-8 portrays array capabilities of the languages considered in this report.

| EXHIBIT 4-8: ARRAY FEATURES | | | | | | |
|---|----------|----------|----------|---------|---------|----------|
| LANGUAGE FEATURE | MUMPS | Ada | C | COBOL | FORTRAN | Pascal |
| Dimensions | No Limit | No Limit | No Limit | 7 | 7 | No Limit |
| Subscript Type | String | Discrete | Integer | Integer | Integer | Discrete |
| Set Lower Bound | N/A | ● | — | — | ● | ● |
| Array Operations: | | | | | | |
| Assignment | — | ● | — | ●* | — | ● |
| Comparison | — | ● | — | ●* | — | — |
| * As character string, not element-by-element | | | | | | |
| Key: ● Yes — No | | | | | | |

MUMPS differs from other languages since it represents arrays as sparse hierarchical trees addressed by character-string subscripts, rather than as rectangular matrices addressed by integers or other scalar values. MUMPS allows operations only on individual elements of an array. Ada and COBOL allow arrays to be compared for equality; Ada, COBOL, and Pascal allow arrays to be assigned. Ada, C, FORTRAN, and Pascal support additional array operations (e.g., addition or multiplication of numeric arrays) by means of library routines. Since MUMPS lacks these facilities, MUMPS programmers or organizations must provide their own routines for this purpose.

Records (also called structures) are mechanisms for grouping logically related data of different types. For languages with many data types, records are useful for assignment, comparison, and parameter passing. Ada, C, COBOL, and Pascal allow record assignment and parameter passing.

while Ada, C, and COBOL allow comparing records for equality. (See exhibit 4-9.) Since MUMPS allows text and numeric data to be represented in a single array, MUMPS arrays are somewhat analogous to records. As noted earlier, however, MUMPS allows operations only on individual elements. Furthermore, since MUMPS lacks scalar data typing, it does not provide automatic range validation on the individual elements that make up a composite. Instead, programmers must perform any checking explicitly in their program code.

| EXHIBIT 4-9: RECORD FEATURES | | | | | | |
|------------------------------|-------|-----|---|-------|----------|--------|
| LANGUAGE FEATURE | MUMPS | Ada | C | COBOL | FORTTRAN | Pascal |
| Internal Records | — | ● | ● | ● | — | ● |
| Record Operations: | | | | | | |
| Assignment: | | | | | | |
| Entire Record | N/A | ● | — | ● | N/A | ● |
| Corresponding Name | N/A | — | — | ● | N/A | — |
| Comparison | N/A | ● | — | ● | N/A | — |
| Key: ● Yes — No | | | | | | |

Sets are unordered collections of elements chosen from a specified domain. Set operations include inserting and deleting elements; testing whether one set is a subset of another; and taking the union, intersection, negation, or difference of two sets. Pascal supports sets directly, while Ada and C support sets somewhat less directly as arrays of logical data and bit strings, respectively. MUMPS, like FORTRAN and COBOL, has no support for sets. This limitation requires MUMPS programmers to choose alternative algorithms that may be less direct and more error prone compared to those available to programmers using languages that support set operations.

4.4 Interfacing

Interfacing describes the capabilities a language offers for communicating with external processes or persons. Interactive support describes facilities for communicating with a computer terminal or similar device; file input/output describes access to organized collections of data on disk, tape, or other media.

4.4.1 INTERACTIVE SUPPORT

All of the languages discussed here provide facilities for displaying text results on a screen and for accepting text input from a keyboard. Typically the language itself provides direct support for the input and output of character data (e.g., the ASCII character set). Support for screen formatting and graphics is typically not standardized, but provided through libraries of modules called device handlers and device drivers. Similarly, control for nontext devices (graphics terminals, pointing devices, "smart" medical instruments, etc.) is generally outside the scope of programming language standards; it may be achieved through bindings to external standards. (See section 4.5.2.)

MUMPS provides facilities for interactive support (terminal I/O) that are comparable and sometimes superior to the other languages considered here. Control for text terminals is built into the MUMPS language and will be enhanced in the proposed 1993 standard. Device control will be further enhanced in the proposed standard through bindings. (See section 4.5.2.)

4.4.2 FILE INPUT/OUTPUT

File input/output is an important mechanism for data storage and for communication among different systems and computers. Files may be internal to a specific language and computer system when data storage is the primary purpose. External files are used to exchange data between potentially diverse systems. (See exhibit 4-10.)

| EXHIBIT 4-10: FILE INPUT/OUTPUT | | | | | | |
|-------------------------------------|-------|----------|----------|---------|---------|----------|
| LANGUAGE FEATURE | MUMPS | Ada | C | COBOL | FORTRAN | Pascal |
| Internal File Aggregate | Text | Any Type | Any Type | Record | List | Any Type |
| External Files: | | | | | | |
| Fixed-Format I/O | No | I/O | I/O | I/O | I/O | Output |
| Free-Format I/O | No | I/O | I/O | Output* | I/O | I/O! |
| * Only for certain hardware devices | | | | | | |
| ! Input works only with numbers | | | | | | |

Internal files typically are composed of some type of data aggregate (such as records), organized in one of a few well-defined formats and encoded for efficient I/O operations and data storage. They are sometimes called "binary" files to distinguish them from character-based (external) files. MUMPS, like Ada, C, COBOL, FORTRAN, and Pascal provides support for internal files. MUMPS internal files are character data represented as sparse hierarchical arrays.

External files are usually strings of characters in either fixed or free format, separated by end-of-line characters or other delimiters. Typically, external files contain atomic data rather than aggregates. Ada, C, and FORTRAN support both free- and fixed-formats for external files.

COBOL supports only fixed-format external files, while Pascal supports only free-format external files.

Standard MUMPS does not support external files; the ability to read and write text is provided by vendor-specific extensions. Programmers can parse incoming files or create formatted text using MUMPS string-handling features. This provides great potential flexibility in reading files created by other programming languages, at the cost of developing custom program logic for reading each file format.

4.5 Extendibility

As noted in section 4.4, the languages discussed here support interfacing with external persons or processes primarily through the use of libraries of specialized modules. In a similar way, libraries and bindings provide mechanisms for extending the capabilities of a programming language into specialized areas.

4.5.1 LIBRARIES

A library is an organized collection of program components, usually functions or subroutines, that can be used as part of a larger program. Typically, the standards for a programming language include a modest set of library routines; these are often called "intrinsic" functions. Additional libraries may be available as a result of DoD software reuse efforts, may be defined by industry consensus, or may be vendor or implementation dependent. Such libraries are generally organized around particular functional areas. For example, libraries of mathematical or statistical operations are common. (Libraries of input/output routines were discussed in section 4.4.) Exhibit 4-11 portrays library capabilities.

The current MUMPS language standard includes functions for string handling, array traversal, and the generation of pseudo-random numbers. The proposed 1993 standard will include functions for exception handling (discussed in section 4.2.1.4), transaction processing, and improved device handling. It also will include (for the first time) a standardized facility for calls

| EXHIBIT 4-11: LIBRARY FEATURES | | | | | | |
|--------------------------------|-------|-----|---|--------|----------|--------|
| LANGUAGE FEATURE | MUMPS | Ada | C | COBOL | FORTTRAN | Pascal |
| Intrinsic Functions | | | | | | |
| String Functions | ● | — | ● | — | ● | — |
| Math Functions | — | — | ● | — | ● | — |
| Type Conversion | N/A | — | ● | — | ● | — |
| Device Control | ● | — | — | Syntax | Partial | — |
| External Libraries | — | ● | ● | ● | ● | — |
| Key: ● Yes — No | | | | | | |

to external subroutines and functions.

In contrast, C and FORTRAN provide standard functions that include extensive string, mathematical, and type-conversion functions. They include some device control and exception-handling functions as well. COBOL and Pascal provide somewhat more restricted libraries of standard functions in these areas. (COBOL provides some of these features by direct syntax rather than through a library.) Similar libraries for Ada are being prepared as a draft standard entitled Ada Generic Package of Elementary Functions (GPEF). Libraries that implement this draft standard are currently available from commercial sources.

Current DoD software reuse activities focus on the Ada language. The DoD has established a DoD Reuse Program Initiative Office to promote and coordinate reuse, and several reuse libraries are available.

An additional level of standardization is provided by the Common Ada Programming Support Environment (APSE) Interface Set (CAIS), MIL-STD-1838A. This set of standardized interfaces is designed to promote the source-level portability of Ada software development tools. It standardizes those tool-to-host interfaces that are most crucial for tool portability.

4.5.2 BINDINGS

Bindings provide a standardized means to link a programming language to other Federal or industry standards for data base access, user interfaces, device control, or other specialized purposes.

The proposed 1993 revision to the MUMPS language standard will include bindings to the MIT X-windows routines to support communications with advanced terminal display devices. It also will include a functional binding to the ANSI X3.64 device control standard. The proposed standard also provides for embedded statements of the Structured Query Language (SQL), and it includes a system-level protocol called Open MUMPS Interconnect (OMI) that will allow information to be shared between MUMPS implementations on different computers.

Similar bindings exist for other languages, especially Ada and C. Standardization levels for such bindings vary; some have been accepted as Institute of Electrical and Electronics Engineers (IEEE), ANSI, FIPS, or ISO/IEC standards, while others are in draft form. A standard for binding Ada to POSIX was accepted by IEEE in June 1992, and approval by ANSI and FIPS is anticipated. Commercial implementations are already available. Standards for binding Ada to SQL, the Graphical Kernel System (GKS), and Programmers Hierarchical Interactive Graphics System (PHIGS) have been approved by ANSI and either FIPS or ISO. Both commercial and Government software to implement these bindings are available. In addition, a draft standard for binding Ada to the Information Resource Dictionary System (IRDS) has been submitted to ISO. Software that implements Ada bindings to Transmission Control Protocol/Internet Protocol (TCP/IP), the X-windows system, and OSF/Motif are available commercially, but standards for these bindings have not yet been drafted.

A binding of C to POSIX has been standardized by IEEE and ISO. Software that implements C language bindings to SQL, X-windows, GKS, PHIGS, TCP/IP, and other standards has been available for some time; in many cases, C was used in the initial implementation of these standards.

Current information is incomplete concerning the status of standards and software implementation for bindings between COBOL, FORTRAN, and Pascal and the standards cited here. In general, standardization and implementation of bindings between these languages lag behind similar efforts for Ada and C.

4.6 Summary of Technical Characteristics

In summary, exhibit 4-12 depicts the most important strengths and weaknesses of each language. Ada is clearly the most powerful language considered here. It incorporates the strong syntactic style and control structures, composite types, and abstract data types that originated in Pascal, while adding robust features for concurrency control. It gains additional strength from external libraries and bindings to POSIX and other standards.

EXHIBIT 4-12: LANGUAGE FEATURE SUMMARY

| LANGUAGE FEATURE | MUMPS | Ada | C | COBOL | FORTRAN | Pascal |
|---------------------|-------|-----|----|-------|---------|--------|
| Syntactic Style | — | ● | ● | ○ | — | ● |
| Execution Control | — | ● | ●* | ●* | ○ | ●* |
| Data Control | ● | ● | ● | ○ | ○ | ● |
| Atomic Data Types | — | ● | ● | ● | ○ | ● |
| User-Defined Types | — | ● | ○ | — | — | ● |
| Composite Types | ○ | ● | ● | ○ | — | ● |
| File I/O | — | ● | ● | ● | ○ | — |
| Libraries | ○ | ● | ● | ○ | ● | — |
| Bindings | ○ | ● | ● | ○ | ○ | ○ |

* Lacks full support for concurrency

Key: ● Strong
○ Average
— Weak

C has similar strengths in syntactic style and control structures, composite and advanced data types, external libraries, and bindings to external standards. It lacks standardized concurrency control and extendible syntax (user-defined data types can be manipulated in functions, not in direct extensions to language syntax).

COBOL has strengths in the business applications domain, especially in its capabilities for data formatting and file I/O, that are unmatched in other languages. Its execution control and atomic data types are further strengths.

The principal strength of MUMPS compared to other languages is its capability to store and manipulate variable-length text. It is handicapped by weak syntax and control structures, limited support for mathematical operations, and the absence of user-defined data types.

Pascal is handicapped by poor support for file I/O and libraries. FORTRAN has a weak syntactic style and does not support records as composite types. It provides no support for user-defined types. Pascal and FORTRAN have been eclipsed by Ada and C, which incorporate and build on their strengths while correcting their weaknesses.

5.0 APPLICATION REQUIREMENTS

Every application is designed to satisfy a particular set of requirements. This chapter discusses how the language characteristics discussed in chapter 3.0 and the technical language features discussed in chapter 4.0 relate to the functional requirements of applications. Functional application requirements are defined by the types and numbers of functions that the application is to perform, such as calculations or data entry. Application requirements are also defined by the number and skill levels of the programmers who will develop the application. Other application requirements are defined by the development process and life cycle of the application, such as whether it will be developed through rapid prototyping and successive refinement or through a more formal system requirements definition and sequential life cycle. Finally, application requirements are defined based on how the application is to be fielded; these requirements may be for performance, reliability, portability, or interfacing.

Exhibit 5-1 shows how the application requirements are related to the evaluation criteria discussed in chapter 3.0. Exhibit 5-2 shows how the application requirements are related to the technical language features discussed in chapter 4.0.

5.1 Functional Support

The most important of all the requirements is that the language provide capabilities that support the functions of the application. For example, if scientific calculation is to be performed, then a language that supports floating-point numbers would be a better choice than one that does not. The degree to which this requirement is satisfied depends on the data types that are supported by the language and the intrinsic functions available in the language. Flexible data types or user-definable data types can provide adequate capabilities if the natural data types for the application do not exist within the language. Similarly, the ability to define functions or use extrinsic functions may provide acceptable support if intrinsic support is inadequate.

5.2 Size and Complexity

Size and complexity requirements are defined by the size and complexity of the application to be developed. The size and complexity of an application may not be directly correlated. The language features that help reduce a single complex task into sets of simpler tasks also serve to break a single large task into smaller tasks that are managed more easily.

EXHIBIT 5-1: APPLICATION REQUIREMENTS AND EVALUATION CRITERIA

| Evaluation Criteria | Functional Support | Size and Complexity | Life Cycle | Skill Level | Many Developers | End-User Interaction | Reliability | Portability | Performance |
|--------------------------|--------------------|---------------------|------------|-------------|-----------------|----------------------|-------------|-------------|-------------|
| Level of Consensus | — | — | ● | ○ | ● | — | — | ● | — |
| Product Availability | — | ○ | ○ | — | — | — | — | ○ | — |
| Completeness | ● | ○ | ○ | ○ | ○ | ○ | ○ | — | — |
| Maturity | — | — | — | — | — | — | — | ○ | — |
| Stability | — | — | — | — | — | — | — | ○ | — |
| De Facto Usage | — | — | — | — | — | — | — | ○ | ○ |
| Problems and Limitations | ○ | ○ | — | ○ | ○ | ○ | ○ | — | — |

Key: ● Strongly related
○ Weakly related
— Not related

EXHIBIT 5-2: APPLICATION REQUIREMENTS AND LANGUAGE FEATURES

| Language Features | Functional Support | Size and Complexity | Life Cycle | Skill Level | Many Developers | End-User Interaction | Reliability | Portability | Performance |
|-----------------------------|--------------------|---------------------|------------|-------------|-----------------|----------------------|-------------|-------------|-------------|
| Syntactic Style | — | — | O | — | ● | — | — | — | — |
| Semantic Structure | ● | ● | ● | ● | ● | — | ● | — | O |
| Data Types and Manipulation | ● | ● | — | — | O | — | O | — | — |
| Interfacing | O | O | — | — | O | O | — | — | — |
| Extendibility | — | O | — | — | — | — | — | — | O |

Key: ● Strongly related
O Weakly related
— Not related

For complex and large programs, the language must support ways to reduce the overall size and complexity of a program by decomposing it into sets of smaller and simpler subprograms. The degree to which a language meets these requirements depends on the extent to which it assists the programmer in modularizing and decomposing algorithms. These requirements are typically met through support of data and functional abstraction, modularity and separate compilability, data scoping, and encapsulation.

However, for small and simple programs, the degree to which the syntax of language minimizes unnecessary "overhead" correlates with the suitability of the language. For such programs, a simple language would be preferable to a more complex one.

5.3 Typical Life Cycle

An application may be designed with a long or short life cycle, or may be planned for ongoing maintenance. For example, a strategic application would normally be designed to have a long life cycle, with possible scheduled upgrades and phased deployment. For such an application with a long life cycle, the ability to structure, modularize, modify, and optimize the program for efficiency may be more important than the ability to develop the program quickly. Conversely, if an initial operating capability with rapid development or demonstration prototypes is required, then ease of writing and regenerating the application would be more important.

Another life cycle consideration is the relative speed of translation and execution of the source code into an executable application. For example, if the language requires excellent execution efficiency, translation speed may be traded for better optimization. Interpreted languages perform translation at run-time, reducing the time required to execute, but may not execute as efficiently. Interpreted languages are better suited to rapid prototyping than to run-time performance. Interpreters have been developed for compiled languages, and compilers have been developed for interpreted languages, so that the developer can have the advantages of both methods. Having both an interpreter and a compiler for a programming language provides the programmer with the advantages of both modes. The programmer can use the interpreter during development for prototyping and debugging, and the compiler for generating optimized release versions.

5.4 Programming Expertise

An application can be developed by professional programmers, casual programmers, or both. Professional programmers are likely to be highly skilled in the language, with support tools and large code libraries to draw from. Casual programmers, for whom programming is a secondary skill, usually lack such resources and are better served by simpler languages. Professional programmers can also be expected to use more sophisticated features of a language. Casual users

can develop expertise if the language is easy to learn. If professional programmers are to be used, then the amount of de facto usage is an important consideration.

Languages designed for professional programmers allow the programmers to apply their greater insight and ability to structure a program. However, these features for professional programmers may also be traps for inexperienced or unwary programmers. Languages designed for casual programmers minimize the number of concepts that the programmer must master.

5.5 Many Developers

An application can be developed by a group of developers or a single person. If more than one person can be expected to work on a particular part of the source code of an application, then the ability for others to understand programs written in the language is important. The ability to structure and modularize programs written in the language is important to help others, in addition to the code's author, understand what the code does. The ability to make the code readable through meaningful names and functional abstraction is also desirable. ADTs and UDTs provide further support by making assumptions explicit among many developers.

5.6 End-User Interaction

The application can be designed to interact with the user in one of three modes: batch, interactive, and real-time. In batch processing, the user starts the application, and the application completes its functions without further intervention or assistance from the user. In interactive processing, the user can interact with the application, and the application must respond to the user's directives. In real-time processing, the application not only must respond to the user's directives, but the application must also synchronize with external entities. Clearly, interactive and real-time processing have greater input/output demands than batch processing.

5.7 Reliability

Applications language may be required to handle critical functions without failures. Exception-handling capabilities can be used to deal with failures in underlying hardware or erroneous input data. Type checking, functional abstraction, data abstraction, and encapsulation help prevent logical errors within the program. Simplicity and software development support aid in writing correct programs; strong typing can also be helpful in detecting some logical errors at compile time. Explicit exception handling can be performed to some extent in most languages; however, strong support for implicit exception handling is desirable.

5.8 Portability

Portability is the ability to move software from one system to another. Portable applications provide independence from specific hardware vendors, allowing the user to migrate the application to the most cost-effective systems available. Portability is strongly influenced by the level of standardization and by the ability to mask hardware or system-specific details from the rest of the system. Portability allows the application to be developed on a platform that is different from that on which it will be deployed. Portability depends strongly on standardization, both syntactic and semantic.

Portability must be designed into a program from the start. If properly implemented, an application written in any of these languages can be ported easily between systems. If portability was not considered in the original design, then the program is likely to be difficult to port to another system, regardless of the language used.

5.9 Performance

System performance of the application may be important to certain classes of applications. Interactive and real-time embedded applications have required minimum levels of performance or responsiveness for acceptable use. The capabilities that support these types of applications might also be used by the programmer to improve performance. Similarly, a simple language that can be optimized easily by a translator, or as directed by the programmer, may also be used to meet performance requirements. Performance usually can be optimized by trading among code complexity, storage space, and execution speed. Performance depends strongly on the capabilities of the underlying hardware and the quality of the interpreter or compiler used. Nevertheless, the ability for the programmer to guide the compiler to an optimized implementation is a clear advantage.

6.0 LANGUAGE SUITABILITY

This chapter discusses the suitability of the programming languages with respect to the functional application requirements defined in chapter 5.0. Every application has a particular set of requirements that must be satisfied. As stated in chapter 1.0, each programming language was designed to fulfill a specific set of requirements, either explicitly stated or implicitly understood, that other programming languages did not fulfill. Thus, for a particular application, one or more of the languages will be better suited than the other languages. Any of the languages could be used to write a particular application. The benefit of being able to use a better matched language may outweigh the reduced training and maintenance costs associated with using only a single language. Exhibit 6-1 shows how the application requirements defined in chapter 5.0 are addressed by each of the programming languages evaluated.

| EXHIBIT 6-1: LANGUAGE VS. HOL REQUIREMENTS | | | | | | |
|--|--|--------|----------|--------|----------|----------|
| APPLICATION REQUIREMENTS | MUMPS | Ada | C | COBOL | FORTTRAN | Pascal |
| Functional Area: | | | | | | |
| • Business | ● | ○ | ○ | ● | — | — |
| • Math, Science | — | ○ | ● | — | ● | ○ |
| • Systems | — | ○ | ● | — | — | — |
| • Real-time | ○ | ● | — | — | — | — |
| • Education | ○ | — | — | — | ○ | ● |
| Size and Complexity | ○ | ● | ● | ○ | ○ | ● |
| Timeframe | Short | Long | Moderate | Long | Short | Long |
| Expertise | Casual | Expert | Moderate | Expert | Casual | Moderate |
| Many Programmers | ○ | ● | — | ● | — | — |
| End-User Interaction | ● | ● | ○ | — | ○ | ○ |
| Reliability | — | ● | — | — | — | ○ |
| Portability | ● | ● | ○ | ● | ● | ● |
| Performance | ○ | ○ | ● | — | ● | ○ |
| Key: | ● Strong support ○ Moderate support — Weak support | | | | | |

6.1 Functional Support

The functions directly supported by a programming language determine its suitability to a particular application. These are most directly related to the data types and functions directly supported by the language. In languages that support data and functional abstraction, a non-native type can be simulated. For example, fixed-point math, which is useful in business applications, can be simulated with integer math. Other functional areas include scientific and engineering applications, systems programming, and embedded systems.

Business applications require fixed-decimal math, character and string formatting, and file-handling capabilities. COBOL was expressly designed for these requirements; it has the vast majority of installed code. Ada's built-in support for fixed-decimal math, text input/output, and string libraries means that Ada also would be suitable. C's text input/output capabilities and widely available libraries support C for business applications. MUMPS has an integrated data base, and would also be acceptable for business applications.

Scientific, mathematical, and engineering applications require support for manipulation of arrays and matrices of extended precision floating-point numbers. FORTRAN was originally designed for these requirements and has high de facto usage for scientific and mathematical programming. C also provides support for efficient array and matrix calculations. Software libraries of mathematical functions are currently available for C and FORTRAN. Ada also provides adequate support for scientific programming. MUMPS does not support mathematical or scientific applications efficiently, and the MUMPS portability standard places a relatively low limit on the mathematical precision available.

Systems programming must allow the programmer direct access to the underlying hardware. C is the only language with direct support for bit-level operations. It is thus especially well suited to systems programming. Ada provides adequate support for systems programming. MUMPS does not support hardware-level programming directly, and most implementations cannot be compiled to produce a stand-alone operating system.

Embedded applications (excluding systems programming) require support for very large programs, tasking, and exception handling. Ada was expressly designed for real-time and embedded applications and is very well suited for such applications. MUMPS was designed for interactive applications and provides adequate support for such applications.

6.2 Size and Complexity

Large or complex applications require capabilities to reduce them to sets of smaller, simpler tasks. In addition, large and complex applications require that the program structure be readily visible. Ada was designed to support development of large programs and is very suitable for

developing such applications. C, and to a lesser extent COBOL, are also well suited to dealing with large programs. MUMPS provides adequate support for large programs.

Small and simple applications require that the language be simple and easy to understand, with minimal syntactic and conceptual overhead. FORTRAN and MUMPS are well suited for writing such applications. Pascal provides adequate support for such applications.

6.3 Typical Life Cycle

Applications are designed with both long and short life cycles. Applications with long life cycles or extended maintenance requirements require a HOL with strong capabilities to structure, modularize, and document the code. For such systems, Ada, COBOL, and Pascal would be appropriate choices due to their emphasis on declaration and definition. However, applications with short life cycles or rapid development requirements require an HOL with a minimum of syntactic overhead and ease of writing. For such systems, MUMPS provides excellent support for rapid prototyping. C and FORTRAN provide adequate support for rapid prototyping.

MUMPS is the only language reviewed in this paper that is typically interpreted, not compiled. MUMPS can therefore be expected to perform somewhat less efficiently at computation-intensive tasks than an HOL with an optimizing compiler. MUMPS has a number of features, such as indirection and execution of data as code, that make compilation difficult or impossible. On the other hand, interpreters have been developed to assist in debugging HOLs that are typically compiled.

6.4 Programming Expertise

Languages designed for professional programmers allow the programmers to use advanced resources in developing a program. Ada and COBOL are best suited for use by professional programmers with libraries of code from which they can build. Languages designed for casual programmers place minimal requirements on the programmer. MUMPS and FORTRAN are best suited for use by casual programmers with their relative lack of declarations and smaller requirements for effective use of the language. Pascal and C are intermediate cases. C provides strong support for libraries, advanced programming techniques, and many idioms often used incorrectly by inexperienced users; however, C also has a small working set of simple features and low syntactic overhead. Pascal is a more sophisticated language, with a heavy emphasis on declarations, but it lacks the library support and separate compilability features that professional programmers demand.

6.5 Many Programmers

If multiple programmers are expected to develop and maintain the code of an application, then it is important that the language be easy to understand. Standardization is most important; program structure and development support are also important. Ada and COBOL were both designed to be easy for others to understand. Ada provides excellent support for multiple programmers, with strong facilities for structuring and modularizing code. COBOL and Pascal provide good support for multiple programmers, with their emphasis on declarations and structure. MUMPS provides adequate support for multiple programmers through enforced modularization and the conventional usage of name spaces.

6.6 End-User Interaction

All languages would be acceptable for batch mode processing; all have some file input/output capabilities. However, MUMPS does not support batch mode processing as well as the other languages do, since it was designed for interactive processing.

For interactive processing, Ada, C, and MUMPS are the strongest, although the other languages would provide adequate support. COBOL and Pascal are relatively weak at handling free-form input/output. As noted below, performance may also be an issue for effective interactive processing.

For real-time processing, Ada clearly has the strongest capabilities. Ada was designed for real-time embedded systems, and has very powerful support for concurrency control. MUMPS has limited concurrency capabilities that stem from its native mode implementations on the DEC PDP-11, in which MUMPS handled operating system functions. C has limited real-time processing capabilities that are drawn from its use in writing operating systems.

6.7 Reliability

Some languages have features that directly support the creation and maintenance of reliable applications; most of the languages evaluated lack such features. Ada provides excellent support for writing reliable programs, with exception-handling capabilities, strong type checking, functional and data abstraction, and encapsulation. Because it is strongly typed, Pascal provides good support for writing reliable programs, with abstraction capabilities that are almost as strong as with Ada. MUMPS provides relatively weak support for writing reliable programs. It is a weakly typed language that has only recently added capabilities for functional abstraction, structured programming, and encapsulation; MUMPS error handling is being standardized.

6.8 Portability

Portability is the ability to move software from one system to another. Portability must be designed into a program from the start. If properly implemented, an application written in any of these languages can be ported easily between systems. If portability was not considered in the original design, then the program is likely to be difficult to port to another system. Ada and MUMPS have excellent portability; all Ada compilers must be certified and the MUMPS portability standard defines semantic requirements for portable programs. COBOL, FORTRAN, and Pascal have good portability due to their strong level of standardization. Because C has only recently been standardized at the national and Federal level, it has only average portability.

6.9 Performance

System performance of the application may be important to certain classes of applications. Interactive, real-time, and embedded systems applications may require minimum levels of performance for acceptable use. Sophisticated languages such as Ada, C, and Pascal allow the programmer to incorporate higher performance routines or to optimize the code at compile time. C, FORTRAN, and Pascal all have been designed to produce very efficient implementations when compiled.

7.0 CONCLUSION

This chapter summarizes the MUMPS evaluation with respect to the DoD documentation and criteria. For comparative purposes, other languages that met criteria defined in chapter 1.0 also were examined. For each of those languages, the following characteristics are summarized:

- Support for large programs
- Strengths and weaknesses of the language
- Functional suitability of the language
- Business and other considerations

7.1 MUMPS

The MUMPS programming language provides moderate support for large programs through library support, functional abstraction, and modular and structured code. The conventional usage of "name spaces," block structuring, and subroutines helps to further support modular software applications. The lack of conventional variable scoping, strong data typing, and declarative statements makes MUMPS less than ideal for large-scale software development projects.

MUMPS is quite weak at mathematical and statistical calculations due to its lack of matrix arrays and inability to store and read data in binary form.

The MUMPS programming language is strong in terms of string handling and text data management. The language includes powerful string manipulation operators and is best suited for managing data bases of shared variable-length text data. In addition, MUMPS is moderately well suited for rapid prototyping, since programs can be modified as they are running, and the results of changes can be seen immediately.

The weaknesses of MUMPS in comparison with other Higher Order Languages are counter balanced by its emphasis on string operations, text data management, and efficient use of machine resources and coding effort (efficient data storage, low memory requirements, terse syntax, and flexibility). Nonetheless, MUMPS is inferior to other Higher Order Languages in data typing, mathematical and statistical functions, execution control structures, and syntactical style. Because of these limitations, MUMPS is not as adaptable in supporting a wide range of requirements and functional areas. In comparison with FORTRAN, MUMPS is superior in character-string operations while FORTRAN is far superior in mathematical and scientific functions. COBOL provides much stronger data typing and file I/O, while MUMPS is clearly superior in storing and manipulating variable-length text data.

MUMPS portability is as good as or better than any of the other languages discussed here. MUMPS currently has significant limitations in support for libraries and bindings to industry or Federal standards. The proposed 1993 standard will address some of these limitations. MUMPS is not widely known, but has a small community of dedicated supporters.

7.2 Ada

Ada was designed with particular attention to the challenges in developing very large programs, including those in other languages. Ada's capabilities for modular and structured code, strong typing, and advanced data types make it the language of choice for large programs.

Ada's data design also emphasizes machine independence. Ada programmers can define data types in terms of the required precision without being restricted to representation on a particular hardware platform. The scope of data is clearly and consistently defined. Further, Ada provides complete and consistent concurrency features. This makes Ada the language of choice for applications that require concurrency.

Further Ada advantages are its libraries of reusable modules and the availability of bindings to Federal and industry standards. Many of these have been developed under Federal auspices, and are available without license fees or restrictions.

Although Ada is used to only a modest degree in the commercial sector, its use is increasing. Ada is becoming more widely used in academia.

7.3 C

C is generally suitable for large programs due to its strong support for separately compiled libraries, block structure, and modular code. Clear rules for the scope of variables also contribute to its success for large systems.

C has both power and flexibility; bit-level and register operations provide the machine interaction that is desirable for operating systems implementation at the potential expense of reduced portability. Because of its historical association with the UNIX operating system, libraries are available to support system calls. Device control and concurrency are provided through this mechanism.

Many of the widely used bindings to Federal and industry standards (GKS graphics, X-windows, terminal interface, etc.) were initially developed in C. In addition, many commercial libraries are available for engineering, graphics, and other areas.

C is used widely in commercial software development, and is challenging FORTRAN in scientific and engineering applications. C use is growing rapidly in academia.

7.4 COBOL

COBOL is suitable for large programs within business information systems, its intended domain. It supports modular and structured code. COBOL's limitations on data scoping make it somewhat cumbersome on larger projects.

COBOL's main strength is in business processing; it strongly supports file input/output, fixed-precision decimal arithmetic, and reporting.

COBOL is unsuitable for intensive mathematical operations and has no concurrency facilities. Its support for interactive input/output and graphics is comparatively weak.

COBOL is widely used for business data processing in both the commercial and Government sectors. It has commercial bindings to SQL data bases and to proprietary transaction processing software.

7.5 FORTRAN

FORTRAN is moderately suitable for large programs. FORTRAN modules can be compiled separately, and many commercial libraries are available.

FORTRAN was the first major language to support high-precision numeric data types and multi-dimensional matrices. Its modular structure enabled the creation of the first important libraries of reusable modules. Limitations on variable name size and data scoping are significant, but they have been overcome on many projects.

FORTRAN continues to be appropriate for computation-intensive applications in science and engineering. FORTRAN remains less well suited toward text-based string manipulation.

While it has long been the de facto standard for scientific and engineering applications, FORTRAN is losing ground to C. This is attributable in part to the association of C with the UNIX operating system, and to the growing number of tools that support C programming.

7.6 Pascal

ANSI Standard Pascal is severely constrained for large systems, but many successful large systems have been created using Pascal with extensions. ANSI Standard Pascal is hampered by the lack of standardized separate compilation and library support; each program must be entirely contained within a single file.

Pascal supports block structuring and modular code, and strong data typing and clearly defined scope. It supports user-defined data types and provides type checking for them. Success with these Pascal features led to their inclusion in Ada.

Pascal continues to be appropriate for applications that require numeric computation and substantial data handling.

Pascal has been highly successful as a teaching language, and has had some use in commercial systems. It is being largely eclipsed by C in the commercial sector, and by Ada in the Government sector and academia.

7.7 Observations

Although each language is well suited for its original purpose, some languages offer more capabilities than others. MUMPS is well suited for business and data base applications. Ada is especially well suited for real-time embedded systems and is reasonably well suited for business, scientific, and systems programming. C is well suited to business, scientific, and systems programming. COBOL is suitable only for business programming. FORTRAN is best suited for scientific programming. Ada and C have sufficient flexibility to perform the functions of COBOL, FORTRAN, and Pascal. MUMPS appears to be competitive with COBOL for business programming.

Ada and C appear to be the dominant languages for the near term. Ada is strongly backed by DoD, which is creating tools and services to support development in Ada. C is strongly backed by commercial and other users, to the extent that most language bindings are first developed for C and then extended to other languages such as Ada or MUMPS.

Overall, MUMPS appears to meet the minimum DoD criteria for Higher Order Languages nearly as well as do COBOL, FORTRAN, and Pascal. However, as HOLs, Ada and C are clearly superior to MUMPS. COBOL, FORTRAN, and Pascal also rate somewhat higher as HOLs. MUMPS is a suitable alternative for AISs that are well matched with the strengths of the language. There are several domains of AISs where MUMPS is not an ideal alternative such as large-scale development projects, AISs requiring tight integration with existing non-MUMPS-based systems, AISs not well matched with the strengths of the language, and AISs requiring

comprehensive support for DoD and industry Standards. To the extent that it continues to evolve toward open systems and standards, MUMPS will remain viable for near-term use. The fact that MUMPS is changing to meet those requirements bodes well for the language.

APPENDIXES

APPENDIX A REFERENCES

A.1 Standards and Official Documents

- ANSI/IEEE770X3.97-1983 American National Standard Pascal Programming Language.* American National Standards Institute. 1983.
- ANSI/MDC X11.1-1990 MUMPS Language Standard.* MUMPS Development Committee. 1990.
- ANSI/MIL-STD-1815A-1983 American National Standard for the Ada Programming Language.* American National Standards Institute. 1983.
- ANSI/X3.9-1978 Programming Language FORTRAN.* American National Standards Institute. 1978.
- ANSI/X3.23-1985 Programming Language COBOL.* American National Standards Institute. 1985
- Application Portability Profile: The U.S. Government's Open System Environment Profile - OSE/I Version.* National Institute of Standards and Technology. Draft, 27 October 1992.
- "Computer Programming Language Policy." DoD Directive 3405.1. 2 April 1987.
- "Delegation of Authority and Clarifying Guidance on Waivers from the Use of the Ada Programming Language." Assistant Secretary of Defense: Command, Control, Communications, and Intelligence. 17 April 1992.
- Dictionary for Information Systems.* American National Standards Institute Standard X3.172-1990. 1990.
- "Glossary of Software Engineering Terminology." IEEE Standard 610.12-1990. *IEEE Software Engineering Standards Collection.* Spring 1991.
- Selection and Use of General-Purpose Programming Languages - Overview.* U.S. Department of Commerce, NBS Special Publication 500-117 Volume 1. October 1984.

A.2 Additional References

- ABCs of MUMPS: An Introduction for Novice and Intermediate Programmers.* Richard F. Walters, Digital Press. 1989.
- Ada From the Beginning.* J. Skansholm, Addison-Wesley. 1988.
- Ada: An Introduction.* H. Ledgard, Springer-Verlag. 1981.
- "An Analysis of the Impact of the Argumentless DO on the MUMPS Programming Language." B. Isch. *MUMPS Computing*: Volume 22, Number 3. 1992.
- The C Programming Language* (Second edition). B.W. Kernighan and D.M. Richie, Prentice-Hall. 1988.
- The Complete MUMPS: An Introduction and Reference Manual for the MUMPS Programming Language.* John Lewkowicz, Prentice-Hall. 1989.
- Concepts of Programming Languages.* R.W. Sebesta, Benjamin/Cummings. 1989.
- "MUMPS." William J. Harvey and Frederick G. Kohun. Unpublished draft 8 for *Encyclopedia of Microcomputers*. 1991.
- "MUMPS: Characteristics and Comparisons with Other Programming Systems." T. Munnecke, R.F. Walters, J. Bowie, C.B. Lazarus, and D.A. Bridger. *Medical Informatics*: Volume 2, Number 3. 1977.
- Pascal User Manual and Report* (Third edition). N. Wirth, Prentice-Hall. 1985.
- Programming Language Concepts and Paradigms.* D.A. Watt, Prentice-Hall. 1991.
- Programming Languages: Design and Implementation* (Second edition). T.W. Pratt, Prentice-Hall. 1984.
- "System Performance: A Benchmark Study of MUMPS and Other Systems." C.M. Alonzo. *MUG Quarterly*: Volume 18, Number 3/4. 1988.
- "What's in the Next Standard." Thomas Salander, unpublished manuscript. October 1992.

APPENDIX B

ACRONYM DEFINITIONS

| | |
|----------|---|
| ADT | Abstract Data Type |
| ANS | American National Standard |
| ANSI | American National Standards Institute |
| APP | Application Portability Profile |
| APSE | Ada Programming Support Environment |
| AST | Advanced Software Technology |
| CAIS A | Common APSE Interface Set Revision - A |
| CIVC | CAIS A Implementation Validation Capability |
| C3I | Command, Control, Communications, and Intelligence |
| DBMS | Data Base Management System |
| DDI | Director of Defense Information |
| DoD | Department of Defense |
| DoDD | Department of Defense Directive |
| FIPS | Federal Information Processing Standard |
| FIPS PUB | Federal Information Processing Standard Publication |
| GKS | Graphical Kernel Services |
| GPEF | Generic Package of Elementary Functions |
| GUI | Graphical User Interface |
| HOL | Higher Order Language |
| I-CASE | Integrated Computer-Aided Software Engineering |
| I/O | Input/Output |
| IEC | International Electrotechnical Committee |
| IEEE | Institute of Electrical and Electronics Engineers |
| IHS | Indian Health Service |
| IRDS | Information Resource Directory System |
| ISO | International Organization for Standardization |
| MDC | MUMPS Development Committee |
| MIL-STD | Military Standard |
| MUG | MUMPS Users Group |
| MUMPS | Massachusetts General Hospital Utility Multi-Programming System |
| NIST | National Institute of Standards and Technology |
| OMI | Open MUMPS Interconnect |
| OSE | Open Systems Environment |
| PHIGS | Programmers Hierarchical Interactive Graphics System |
| POSIX | Portable Operating System Interface |
| PSE | Programming Support Environment |
| RDBMS | Relational Database Management System |
| SQL | Structured Query Language |

| | |
|--------|---|
| TCP/IP | Transmission Control Protocol/Internet Protocol |
| TRM | Technical Reference Model |
| UDT | User-Defined Type |
| VA | Department of Veterans Affairs |

**ANALYSIS OF MUMPS AS AN
ADVANCED SOFTWARE TECHNOLOGY (AST)**

TABLE OF CONTENTS

| | Page |
|---|------|
| LIST OF EXHIBITS | iv |
| EXECUTIVE SUMMARY. | ES-1 |
| 1.0 INTRODUCTION | 1-1 |
| 1.1 Study Background | 1-1 |
| 1.2 MUMPS Background | 1-2 |
| 1.3 DOD Standards Policy | 1-2 |
| 1.4 Data Sources | 1-2 |
| 1.5 Document Organization | 1-3 |
| 2.0 AST CONTEXT | 2-1 |
| 2.1 AST Applied to Software Engineering | 2-1 |
| 2.1.1 IEEE Definitions | 2-1 |
| 2.1.2 Other Definitions | 2-2 |
| 2.2 Additional Criteria | 2-3 |
| 2.2.1 Flexibility | 2-4 |
| 2.2.2 Efficiency | 2-4 |
| 2.2.3 Extendability | 2-4 |
| 2.2.4 Completeness | 2-4 |
| 2.2.5 Maturity | 2-4 |
| 2.2.6 Stability | 2-4 |
| 3.0 MUMPS BACKGROUND | 3-1 |
| 3.1 Distinctive Characteristics of Mumps | 3-1 |
| 3.1.1 MUMPS as a Programming Language | 3-2 |
| 3.1.2 MUMPS as a Database Management System | 3-5 |
| 3.2 MUMPS Standardization | 3-8 |
| 3.2.1 The MUMPS Development Committee | 3-8 |
| 3.2.2 MUMPS Standardization Process | 3-9 |
| 4.0 MUMPS AND AST | 4-1 |
| 4.1 MUMPS Programming Language | 4-1 |
| 4.1.1 Interpretive Nature of MUMPS | 4-1 |
| 4.1.2 MUMPS Syntax | 4-3 |
| 4.1.3 Data Types | 4-4 |

TABLE OF CONTENTS (Continued)

| | | |
|--------|--|------|
| 4.1.4 | String Manipulation Operators and Functions | 4-4 |
| 4.1.5 | Mathematical/Statistical Operators and Functions | 4-4 |
| 4.1.6 | Program Control Structures | 4-5 |
| 4.1.7 | Program Routine Library Support | 4-8 |
| 4.1.8 | Intraprocess Communications | 4-8 |
| 4.2 | Database Management System | 4-9 |
| 4.2.1 | Data Structures and Storage | 4-9 |
| 4.2.2 | Data Retrieval | 4-10 |
| 4.2.3 | Data Integrity/Security | 4-10 |
| 4.2.4 | Data Integration | 4-11 |
| 4.3 | Programming Support Environment | 4-12 |
| 4.3.1 | Specification/Design Phase | 4-12 |
| 4.3.2 | Edit/Link/Test Cycle | 4-12 |
| 4.3.3 | Configuration Management | 4-13 |
| 5.0 | SUMMARY | 5-1 |
| 5.1 | AST Category of MUMPS | 5-1 |
| 5.1.1 | Software Tool | 5-1 |
| 5.1.2 | Life Cycle Support Environment | 5-1 |
| 5.1.3 | Programming Support Environment | 5-2 |
| 5.1.4 | Nonprocedural Language | 5-2 |
| 5.1.5 | Modern Database Management System | 5-3 |
| 5.1.6 | Other Technology | 5-3 |
| 5.2 | AST Capabilities of MUMPS | 5-3 |
| 5.2.1 | Productivity | 5-5 |
| 5.2.2 | Usability | 5-5 |
| 5.2.3 | Maintainability | 5-6 |
| 5.2.4 | Portability | 5-6 |
| 5.2.5 | Flexibility | 5-6 |
| 5.2.6 | Efficiency | 5-7 |
| 5.2.7 | Extendability | 5-7 |
| 5.2.8 | Completeness | 5-7 |
| 5.2.9 | Maturity | 5-8 |
| 5.2.10 | Stability | 5-8 |
| 5.3 | Conclusions | 5-9 |

TABLE OF CONTENTS
(Continued)

| | |
|------------------------------|-----|
| APPENDIX | A-1 |
| A.1: Abbreviations | A-1 |
| A.2: References | A-3 |

LIST OF EXHIBITS

| No. | | Page |
|------|--|------|
| ES-1 | AST Capabilities | ES-2 |
| ES-2 | MUMPS AST CRITERIA ASSESSMENTS | ES-3 |
| 3-1 | Distinctive Characteristics of MUMPS | 3-1 |
| 3-2 | MUMPS Symbolic Operators | 3-3 |
| 3-3 | One-Character Commands in MUMPS | 3-4 |
| 3-4 | The Tree Structure of MUMPS Arrays | 3-7 |
| 3-5 | Timeline of MUMPS Standardization | 3-9 |
| 3-6 | The MUMPS Standards Process | 3-10 |
| 4-1 | AST Capabilities | 4-2 |
| 4-2 | Example of IF/ELSE Flow Control | 4-6 |
| 4-3 | Example of Block Structuring and Parameter Passing | 4-7 |
| 5-1 | MUMPS AST Criteria Assessments | 5-4 |

EXECUTIVE SUMMARY

Department of Defense (DoD) policy, as stated in DoDD Directive (DoDD) 3405.1, Computer Programming Language Policy, is to construct software using the following tools, in descending order of preference:

- Off-the-shelf application packages and Advanced Software Technology (AST)
- Ada-based software and tools
- Approved standard Higher Order Languages (HOLs)

Much of the software developed for the Military Health Services System has been created using the Massachusetts General Hospital Utility Multi-Programming System (MUMPS). This paper evaluates MUMPS against the criteria defined for an AST. MUMPS is evaluated based on two standards: the 1990 American National Standard (ANS), which is the most recently approved, and a proposed ANS for which approval is expected in 1993.

In a memorandum entitled "Delegations of Authority and Clarifying Guidance on Waivers from the Use of the Ada Programming Language, dated 17 April 1992, the Assistant Secretary of Defense (Command, Control, Communication, and Intelligence) defined AST as follows:

Software tools, life-cycle support environments (including programming support environments), non-procedural languages, modern database management systems, and other technologies that provide significant improvements in productivity, usability, maintainability, portability, etc., over those capabilities commonly in use.

In addition to the four criteria listed above (productivity, usability, maintainability, and portability), MUMPS was also assessed on the criteria of flexibility, efficiency, extendability, completeness, maturity, and stability. These criteria came from the Application Portability Profile of the National Institute of Standards and Technology (NIST) and the Standard Glossary of Software Engineering Terminology published by the Institute of Electrical and Electronics Engineers (IEEE). These additional criteria are assumed to be covered by the term "etc." in the AST definition.

MUMPS is in many ways a unique system in that it integrates capabilities normally provided separately by programming languages, database management systems, and programming support environments. Each of these classes of capability was assessed the AST criteria on a scale of poor, fair, average, good, and excellent. Exhibit ES-1 shows that these individual assessments covered the entire scale. As a programming language, for example, MUMPS has poor mathematical and statistical capabilities when compared against most relevant criteria, but it provides excellent portability and flexibility in handling data types

Based on the individual assessment of capabilities, MUMPS then was assessed overall against

EXHIBIT ES-1: AST CAPABILITIES

| Feature | Productivity | Usability | Maintainability | Portability | Flexibility | Efficiency | Extendibility | Completeness | Maturity | Stability |
|--------------------------|--------------|-----------|-----------------|-------------|-------------|------------|---------------|--------------|----------|-----------|
| Interpreted Language | Good | Good | Good | n/a | Good | Average | n/a | n/a | n/a | n/a |
| Syntax | Good | Average | Poor | n/a | n/a | Good | Fair | n/a | n/a | n/a |
| Data Types | Good | n/a | Fair | Excellent | Excellent | Good | Good | Fair | n/a | n/a |
| String Functions | Good | Excellent | n/a | Excellent | n/a | Good | Average | Excellent | n/a | n/a |
| Math/Stat Functions | Poor | Poor | n/a | Poor | n/a | Poor | Good | Poor | n/a | n/a |
| Control Flow | Average | Fair | Fair | n/a | Average | n/a | n/a | Fair | n/a | n/a |
| Library | Fair | Average | Average | Average | Average | Average | Average | Average | n/a | n/a |
| Data/Record Structure | n/a | n/a | Average | n/a | Average | Good | Good | Good | n/a | n/a |
| Data Retrieval | Fair | Fair | Fair | Excellent | n/a | n/a | Good | Average | n/a | n/a |
| Data Integrity | Fair | n/a | n/a | n/a | n/a | n/a | Good | Fair | n/a | n/a |
| Data Integration | Fair | Fair | n/a | Fair | Fair | n/a | n/a | n/a | n/a | n/a |
| Design Phase | Average | Average | n/a | n/a | n/a | n/a | n/a | Fair | n/a | n/a |
| Edit, Build, Test Cycle | Fair | Fair | n/a | Fair | n/a | Good | n/a | Average | Fair | n/a |
| Configuration Management | n/a | n/a | n/a | n/a | n/a | n/a | n/a | n/a | n/a | n/a |

each of the AST criteria. Exhibit ES-2 summarizes these assessments. MUMPS receives an excellent rating on portability, largely due to the existence of a Portability Standard. (Few other software tools have a comparable standard.) MUMPS receives a fair rating on stability, primarily because new standards for MUMPS appear more frequently than for other standardized software tools. MUMPS has average or good ratings on the other criteria.

Although MUMPS can be considered separately as a programming language, database management system, or programming support environment, considering it as an integrated whole is more useful. Considered in this way, MUMPS appears to satisfy the AST criteria when used for small to moderate-sized applications that deal primarily with shared unstructured or textual data. For larger applications or those that deal primarily with structured numerical data, MUMPS is less beneficial.

EXHIBIT ES-2: MUMPS AST CAPABILITY ASSESSMENTS

| <u>AST CRITERION</u> | <u>1990 ANS</u> | <u>1993 ANS</u> |
|----------------------|-----------------|-----------------|
| Productivity | Good | Good |
| Usability | Average | Good |
| Maintainability | Average | Average |
| Portability | Excellent | Excellent |
| Flexibility | Average | Average |
| Efficiency | Good | Good |
| Extendability | Average | Average |
| Completeness | Average | Good |
| Maturity | Average | Average |
| Stability | Fair | Fair |

1.0 INTRODUCTION

The Office of Technical Integration has requested a study of the Massachusetts General Hospital Utility Multi-Programming System (MUMPS). This report focuses on one aspect of the study: whether MUMPS is suitable as an Advanced Software Technology (AST) as defined by the Department of Defense (DoD).

DoD requires the use of Ada or other technologies that meet the definition of an AST for development and major enhancement of information systems. This report documents the development of MUMPS and the status of current and proposed MUMPS standards within the context of AST. The information in this report was gathered from DoD, the MUMPS User's Group (MUG), the MUMPS Development Committee (MDC), and the Department of Commerce National Institute of Standards and Technology (NIST).

The American National Standard (ANS) for MUMPS is in a state of transition. The MUMPS Language Standard is being enhanced to support a more open architecture. Interfaces have been proposed to support Open Systems, transaction processing, and data communications in a 1993 revision of the ANS. The revisions are in the final comment stages; they will be finalized in 1993 or 1994.

1.1 STUDY BACKGROUND

DoD Directive (DoDD) 3405.1, Computer Programming Language Policy, states that DoD policy, based on life cycle costs and impact, favors the use of the following:

- Off-the-shelf application packages and Advanced Software Technology
- Ada-based software and tools
- Approved standard Higher Order Languages (HOLs)

The Assistant Secretary of Defense reinforced this policy in a memorandum dated 17 April 1992. This memorandum, titled "Delegations of Authority and Clarifying Guidance on Waivers from the Use of the Ada Programming Language," required all new development or modification of DoD software to comply with DoDD 3405.1.

The DoD requires a waiver for development of software, or modifications to more than one-third of the existing code, that will not use the Ada programming language. This waiver must be supported by documentation demonstrating that the benefits of the AST definition are met. The definition of Advanced Software Technology, stated in DoDD 3405.1, was clarified in the 17 April 1992 memorandum as follows:

Software tools, life cycle support environments (including programming support environments), nonprocedural languages, modern database management systems, and

other technologies that provide significant improvements in productivity, usability, maintainability, portability, etc., over those capabilities commonly in use.

This study examines the current features and changes planned for MUMPS with respect to the AST definition. MUG and the MDC supplied information about the development and planned changes in MUMPS.

1.2 MUMPS BACKGROUND

MUMPS was initially developed by the Laboratory of Computer Science at Massachusetts General Hospital in 1967. MUMPS is both an interactive programming language and a general-purpose database management system that has been widely used in the development of medical information systems.

The MDC publishes both a MUMPS Language Standard and a MUMPS Portability Standard. The MUMPS Language Standard has been approved as an American National Standard (ANSI/MDC X11.1-1990), a FIPS Publication (FIPS PUB 125), and an International Organization Standards (ISO) Standard (ISO/IEC 11756). The MUMPS Portability Standard allows conforming application code to be truly portable among different MUMPS implementations, operating systems, and hardware platforms.

Although MUMPS was originally developed for minicomputers, MUMPS versions are currently available for mainframes, minicomputers, workstations, and personal computers. MUMPS is best known for its ability to handle unstructured textual information, its data sharing capabilities, and the interpretive nature of the language.

1.3 DOD STANDARDS POLICY

Within DoD, the usage of DoD Standards and Military Standards (MIL-STD) is mandatory wherever they apply. A FIPS PUB should be used if no applicable DoD standard or MIL-STD exists. Otherwise, the usage of an American National Standard is recommended unless it conflicts with an existing FIPS PUB, DoD Standard, or MIL-STD.

1.4 DATA SOURCES

The information in this report was gathered through personal interviews, telephone conversations, and a literature search. Staff members within NIST were interviewed to determine whether an evaluation of MUMPS already had been conducted and, if so, what the result was. Contacts provided by MUG and MDC members were consulted to gather information about the MUMPS programming language and environment. Contacts at Robert Morris College were interviewed to gather additional information about the MUMPS pro-

programming language and environment. The MUMPS Language Standard (ANSI/MDC X11.1-1990) was a source of details about the MUMPS Language Standard and Portability Standard. Other studies that examined MUMPS were examined for relevance to this analysis.

1.5 DOCUMENT ORGANIZATION

The remainder of this document is organized as follows. Chapter 2.0 defines the terms used in the definition of AST more completely. These definitions will form the basis for the analysis of MUMPS in the later chapters. Chapter 3.0 provides background on the features of MUMPS as a language and as a database management system; it also describes the initial efforts to standardize MUMPS. This background will be the basis for explaining the relationship between AST and current and planned standards for MUMPS.

Chapter 4.0 describes the key features of MUMPS and how these features relate to the criteria for AST. It describes both programming language features and database management features. Chapter 5.0 summarizes the previous chapters by examining the criteria and expected benefits of AST and assessing MUMPS against each.

Appendix A.1 lists the abbreviations and acronyms used in this report. Appendix A.2 lists the references used in preparing this report.

2.0 AST CONTEXT

This study provides information on the MUMPS programming language with respect to AST. Department of Defense (DoD) policy states that all non-Ada development and maintenance (exceeding one-third of existing code) must be done under a waiver that documents how the alternative chosen achieves the benefits of AST. Continued usage of the MUMPS programming language in DoD systems may require such a waiver when substantial enhancements are proposed. Determining whether the MUMPS programming language is an AST will be useful in preparing and evaluating such waivers.

As noted in chapter 1.0, DoD Directive 3405.1 defines an AST as follows:

Software tools, life cycle support environments (including programming support environments), nonprocedural languages, modern database management systems, and other technologies that provide significant improvements in productivity, usability, maintainability, portability, etc., over those capabilities commonly in use.

It is necessary to define the terms used in this definition to remove ambiguity about their meanings. Because DoD Directive 3405.1 does not define the terms used in the definition of AST, this chapter defines the terms within the context of software engineering in general, and with respect to programming languages in particular. The proposed changes to the MUMPS programming language can then be evaluated with respect to the AST definition.

2.1 AST APPLIED TO SOFTWARE ENGINEERING

The terms used in the definition of AST require technical definitions that are more precise than those from common usage. The Institute of Electrical and Electronics Engineers (IEEE) defines many of these terms in its *IEEE Standard Glossary of Software Engineering Terminology* (IEEE Standard 610.12-1990). This glossary defines terms in the AST definition as they are currently used within the field of software engineering. Those terms that were not defined in the glossary are defined below based on common practice and usage within industry.

2.1.1 IEEE Definitions

The *IEEE Standard Glossary of Software Engineering Terminology* was approved by ANSI in 1991. The definitions of the following terms used in the definition of AST are taken from the glossary:

- Software Tool
- Programming Support Environment
- Nonprocedural Language
- Usability

- Maintainability
- Portability

2.1.1.1 Software Tool

A software tool is a computer program used in the development, testing, analysis, or maintenance of a program or its documentation. Examples include a comparator, cross-reference generator, decompiler, driver, editor, flowcharter, monitor, test case generator, or timing analyzer.

2.1.1.2 Programming Support Environment

A programming support environment (PSE) is an integrated collection of software tools accessed via a single command language to provide programming support capabilities throughout the software life cycle. The environment typically includes tools for specifying, designing, editing, compiling, loading, testing, configuration management, and project management.

2.1.1.3 Nonprocedural Language

A nonprocedural language is one in which the user states what is to be achieved without having to state specific instructions that the computer must execute in a given sequence.

2.1.1.4 Usability

Usability is the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component.

2.1.1.5 Maintainability

Maintainability is the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment.

2.1.1.6 Portability

Portability is the ease with which a system or component can be transferred from one hardware or software environment to another.

2.1.2 Other Definitions

Some of the terms used in the definition of AST were not defined in the IEEE glossary. Based on their common use within industry, the following terms used in the definition of AST are defined below:

- Life Cycle Support Environment
- Modern Database Management System (DBMS)
- Productivity

2.1.2.1 Life Cycle Support Environment

A life cycle support environment is an integrated set of software tools and utilities designed to support a software product from the time it is conceived until it is no longer used. A life cycle support environment might automate functional and data requirements definition, functional and detailed design, development, testing, release, and maintenance.

2.1.2.2 Modern Database Management System

The *American National Standard for Information Systems — Dictionary for Information Systems* (ANSI X3.172-1990) defines a database management system as an integrated set of computer programs that collectively provide all of the capabilities required for centralized management, organization, and control of access to a database that is shared by many users. A DBMS is further defined as a computer-based system used to establish, make available, and maintain the integrity of a database. It may be invoked by nonprogrammers or by application programs to define, create, revise, retire, interrogate, and process transactions; and to update, back up, recover, validate, secure, and monitor the database.

2.1.2.3 Productivity

Productivity is the ease with which systems, programs, and their component functions and procedures can be developed and written using the language. For example, a simple, commonly used measure of productivity is the number of lines of code written per person in a given amount of time.

2.2 ADDITIONAL CRITERIA

In addition to the criteria directly noted within the AST definition, additional criteria may apply to evaluations of MUMPS as AST. These criteria appear to be covered by the term "etc." in the AST definition. Three such criteria are defined in the IEEE glossary:

- Flexibility
- Efficiency
- Extendability

NIST's Application Portability Profile, the document that describes the Government's Open Systems Environment, defines three additional criteria:

- Completeness
- Maturity
- Stability

2.2.1 Flexibility

Flexibility is the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed.

2.2.2 Efficiency

Efficiency is the degree to which a system or component performs its designated functions with minimum consumption of resources. The IEEE glossary defines two subcategories: execution efficiency, which is concerned with the minimal consumption of time; and storage efficiency, which is concerned with the minimal consumption of available storage.

2.2.3 Extendability

Extendability is the ease with which a system or component can be modified to increase its storage or functional capacity.

2.2.4 Completeness

Completeness is the degree to which a standard defines and covers key features necessary in supporting a specific functional area or service.

2.2.5 Maturity

Maturity is the degree to which a specification and its underlying technologies and concepts are well understood. For example, the specification may be defined by a formal mathematical model or the technology may have been used for many years.

2.2.6 Stability

Stability is the likelihood that the specification will not have many or significant changes within the next 2 years.

3.0 MUMPS BACKGROUND

MUMPS was initially developed by the Laboratory of Computer Science of Massachusetts General Hospital in 1967. The objective of this effort was to develop an interactive, multi-user, minicomputer-based system that would support a hospital's special need to store, share, and manipulate textual data. Many of the characteristics of MUMPS were designed to support and manage the varied and sparse data typical of the medical environment. Through the years, MUMPS has evolved into a general-purpose hierarchical database management system that can support a wide variety of applications, such as scheduling, finance, and inventory.

3.1 DISTINCTIVE CHARACTERISTICS OF MUMPS

MUMPS has several characteristics that differentiate it from other programming languages and database managers. Exhibit 3-1 lists the distinctive characteristics of MUMPS. The following sections discuss implications of these characteristics with respect to MUMPS' roles as programming language and database manager.

EXHIBIT 3-1: DISTINCTIVE CHARACTERISTICS OF MUMPS

- MUMPS is an interpreted language.
- Multiple, abbreviated statements can be included on a single line of program code.
- Shared data is available to other routines without program linkages or data declarations; temporary variable values are not removed from memory when routines terminate.
- MUMPS does not support declaration statements.
- MUMPS supports a persistent, shared database; arrays are hierarchical and require no pre-allocation of memory; data on disk are stored in a sparse, hierarchical array.
- MUMPS variables are physically stored as variable-length character strings; they are converted when needed for numeric calculations.
- MUMPS allows the use of string subscripts as keys to array nodes; data is stored physically in sorted order according to subscript values.

3.1.1 MUMPS AS A PROGRAMMING LANGUAGE

Many of the programming characteristics of MUMPS are distinct from those found in other languages. MUMPS did not evolve out of the mainframe environment. The target platform of the language was a set of shared minicomputers operating in an interactive environment. The following sections describe the characteristics and implications of MUMPS as a programming language.

3.1.1.1 Interpretive Nature of MUMPS

MUMPS programs and source code are interpreted at run time. Source code instructions are converted into machine-level instructions at the time of execution, rather than compiled prior to execution. The interpretive nature of MUMPS facilitates rapid prototyping, testing, debugging, and maintenance.

Rapid prototyping is facilitated because the program can be modified directly at run time. This saves the time required to compile a new version to determine whether it is acceptable.

Debugging is facilitated since the programmer can interact with the program at run time and execute commands directly at the keyboard. The location of the error and the conditions that caused the error are immediately available.

Because MUMPS is an interpreted language, applications written in MUMPS tend to run more slowly than equivalent applications written in a compiled language. In order to execute each command, MUMPS must first translate the source code instructions into machine instructions before executing them. Compiled programming languages perform this translation in advance and store the machine-level instructions as a program file that is executed at run time. Compiling programs improves efficiency since the translation of each statement occurs only once, not each time it is executed. In addition, compilers optimize the instructions to perform faster execution.

In order to reduce the overhead associated with run-time interpretation, most implementations of MUMPS "tokenize" the code by partially compiling and optimizing the source code to improve execution speed. Tokenized MUMPS code still requires the MUMPS interpreter to operate. The dynamic structure of MUMPS at run time makes it nearly impossible to produce fully compiled MUMPS code because the compiler cannot predict the absolute memory addresses, required variable space, and instructions to be executed.

Despite the execution penalty of an interpreted language, MUMPS programs can perform very effectively in dealing with textual data, especially where hierarchical storage is appropriate. A benchmark study by Alonzo reported that MUMPS surpassed certain FORTRAN and COBOL implementations for an application of this type.

3.1.1.2 Language Syntax

The syntax of MUMPS programs is typically terse and cryptic. MUMPS makes extensive use of symbolic operators, as shown in exhibit 3-2.

EXHIBIT 3-2: MUMPS SYMBOLIC OPERATORS

The line of code shown below uses the indicated operators to tell MUMPS to check the variable SPONSORID for a sequence of three numeric digits, and if it finds them, to write the variable value on column 20, skip a line, with the variable inside the message "ID [sponsorid]: is valid.":

```
If SPONSORID?3N Write !,?20,"ID ",SPONSORID," is valid."
```

The operators and their meanings are as follows:

| <u>operator</u> | <u>meaning</u> |
|-----------------|-------------------------------|
| ?3N | look for three numeric digits |
| ! | skip a line |
| ?20 | tab to column 20 |

MUMPS has a line orientation whereby several commands can be entered and executed from the same line of source code. Each line in a MUMPS program may have up to 255 characters. To support page and display width limitations, continuation of a line is indicated by three periods at the start of the next line. Most MUMPS commands can be abbreviated to the first letter of the command. Exhibit 3-3 is an example of MUMPS code using one-character commands, two or more to a line.

Combining multiple abbreviated statements on a line makes code easier to write but harder to read. Many MUMPS developers establish programming conventions and standards to enhance the maintainability of the code through adequate program documentation and proper use of MUMPS syntax.

3.1.1.3 Routine Structure

The MUMPS Portability Standard (described in section 3.2.1) limits the collective size of MUMPS routines and local variables executing in main memory to 4,000 bytes. The actual allowable routine size is usually implementation specific. The allowable size of a routine depends on the operating system and hardware issues such as available memory, buffers, disk caches, and other jobs.

EXHIBIT 3-3: ONE-CHARACTER COMMANDS IN MUMPS

The following MUMPS code fragment creates an array of numbers from 1 to 10, searches the array, and writes the highest and lowest values encountered in the array:

```
For A=1:1:10 Set ARY(A)=A
S HI=0, LO=999999
F A=1:1:10 S:ARY(A)<LO LO=ARY(A) S:ARY(A)>HI HI=ARY(A)
W !,"Highest value = ",HI,!,"Lowest value = ",LO
```

Note the use of one-character command abbreviations:

S for Set
F for For
W for Write

MUMPS reads routines from disk or disk buffers when they are called to execute. Routines are removed from memory upon termination. Routines are swapped between main memory and disk storage as needed. The temporary variable space remains unchanged, allowing other routines access to temporary variables. Program linkage declarations are not supported, since routines in a program are not loaded in advance of execution.

The MUMPS routine structure has several implications for the developer:

- Local variables are available for other routines, eliminating the need to declare the variables explicitly.
- MUMPS does not automatically protect the programmer from unintentionally reusing local variables.
- The programmer must explicitly reinitialize local variables to prevent their unintentional reuse.

3.1.1.4 Declarative Statements

MUMPS does not support declarative statements for constants, data structures, variables, procedures, or functions. The following characteristics contribute to the lack of declarative statements in MUMPS:

- MUMPS is a command-line-oriented language. Declarations are defined at the time the function is required.

- MUMPS does not run in compiled mode, so linkages between separately compiled modules are not necessary.
- MUMPS variables and arrays are automatically available to all routines operating in the current process.
- MUMPS variables are physically stored as variable-length character strings.

The elimination of declarative statements in MUMPS is a major departure from other languages. Although the elimination of declarations may make programs easier to write, the use of declarations helps to define clearly the roles of variables and procedures in a program. Use of declarations can also promote early detection of typographic errors. The absence of declarations can reduce maintainability, since declarative statements help to document a program.

3.1.2 MUMPS AS A DATABASE MANAGEMENT SYSTEM

MUMPS was originally designed as both a general-purpose programming language and a DBMS for Medical Information Systems. The principal function of a Medical Information System is to store, retrieve, and manipulate shared textual data. Because of its integrated database system, MUMPS stores and manipulates data differently from other programming languages.

3.1.2.1 File Structures

The MUMPS file structure is a sparse hierarchical array implemented in a tree structure. MUMPS arrays are persistent; that is, they exist even after the routine that created them has stopped running. Local variables are shared by all applications using the same directory. MUMPS provides constructs that the programmer can use to prevent the reuse of local variables.

Most programming languages implement a sequential file structure in the form of a matrix. The matrix structure provides advantages in performing advanced statistical and mathematical functions. These file structures are usually static and hard to adapt to changing requirements. Selecting individual records from a MUMPS hierarchical file is very efficient compared to sequential matrix files.

3.1.2.2 Data Types

Most DBMSs support certain basic data types, such as integer and character, and abstract data types, such as date, time, and currency. MUMPS does not support any data types explicitly. All data is physically stored as variable-length character strings. MUMPS automatically evaluates each variable according to the operation being performed. A numeric operation will return a numeric data type; a character operation, such as a concatenation, will return a character string type. MUMPS will evaluate a variable as either a character string, integer, floating point number, or Boolean value depending on how it is to be used.

Because numbers are handled as text strings, they must be converted into an internal numerical form when used in calculations. These conversions are not necessary for languages that support integer and floating point numbers. Thus MUMPS is much less efficient at complex numerical calculations than a language that supports machine numbers. In addition, MUMPS requires additional space to store numbers since it does not use packed machine-level representations.

MUMPS is essentially a nontyped language. A programmer has an unlimited set of operations that may be performed on data. However, data typing is useful in documenting the nature of an element, domains of its acceptable values, and logical operations that can be performed on it.

3.1.2.3 Arrays

Arrays (referred to in MUMPS as "globals") have no predetermined fixed or allocated size. Records in an array are variable length. Variable names may have up to 31 characters, allowing the programmer to describe the data completely; however, MUMPS uses only the first 8 characters to distinguish between variables.

Most languages implement a matrix array structure with a predefined number of columns and rows. The variable-length record structure of a MUMPS sparse hierarchical array makes efficient use of storage space. Disk storage requirements are determined by the amount of data physically stored, not by a predefined array size and record length; null values and redundant trailing spaces are not saved.

The branches in the hierarchical array structure are termed "subscripts." Subscripts are the keys to the data contained in each node in the array. Each subscript identifies a branch in the array, corresponding to a level in the array hierarchy. The physical ordering of data in an array is determined by the subscript values. Instead of storing data in the order they are entered, MUMPS stores data in sorted order according to the subscript values. Subscript values may be integers or character strings of up to 31 characters.

Storing in order by subscript values improves the search and retrieval ability of MUMPS. For example, a scan of records with the last name starting with the letter "G" will begin sequentially with the first record beginning with "G", terminating when it encounters a name beginning with the letter "H".

MUMPS arrays and subscripts are defined at the time a function uses the array. Exhibit 3-4 illustrates the hierarchical array structure of MUMPS.

The MUMPS hierarchical array structure preceded the relational table model. As in the relational model used in Relational Database Management Systems (RDBMSs), the data is normalized and redundant data is not stored within the system. Tables and attributes in RDBMSs are explicitly defined in advance of their use; in MUMPS, the "tables" and attributes are implicitly defined when they are used. The MUMPS sparse hierarchical structure is most

EXHIBIT 3-4: THE TREE STRUCTURE OF MUMPS ARRAYS

| <u>ARRAY</u> | <u>FIRST SUBSCRIPT (SSN)</u> | <u>SECOND SUBSCRIPT (DATE)</u> | <u>VALUE</u> |
|---------------|--------------------------------------|--|---------------------|
| PATREC | 372-40-1234 | | |
| | | | |
| | 372-40-1349 | | "John Smith" |
| | | 12/01/91 | |
| | | 01/01/92 | "Ultrasound" |
| | | 01/23/92 | |
| | 372-40-2299 | | |
| | | | |

The following statements assign the values indicated in bold above:

```
Set ^PATREC(372-40-1349)="John Smith"
Set ^PATREC(372-40-1349,920101)="Ultrasound"
```

useful for storing textual or other data whose records are of varying length. User views of the data in a RDBMS must be explicitly defined through the relationships between table values; however, hierarchical databases implicitly contain relationship information that indicated that one record is dependent upon another.

Unlike the relational model, entity relationships are not established by matching values in key fields; instead, they are expressed through the hierarchy of the tree. This convention has the following implications:

- A relational database is more flexible than a hierarchical database, since tables can be related to each other in arbitrary ways, as opposed to only through dependency.
- A hierarchical database is much faster to use for certain applications, since the records are implicitly related and two sets of data do not have to be inspected and compared to locate matching values.

Different user views of a MUMPS array would require the view to be defined at the physical level, storing an additional array with redundant subscripts and nodes. The inability to support

different user views from the same physical structure is a significant drawback. If many user views are necessary, the storage efficiency of variable-length record structures is largely negated by the additional redundant data.

3.2 MUMPS STANDARDIZATION

By the early 1970s, several incompatible dialects of MUMPS had evolved. The widespread incompatibility severely hampered the portability of MUMPS applications between environments. The MUMPS Development Committee was subsequently established to define a MUMPS Language Standard to be submitted to the American National Standards Institute (ANSI). The MDC is responsible for revisions to the MUMPS Language Standard and evaluates proposals to modify or enhance the language.

3.2.1 THE MUMPS DEVELOPMENT COMMITTEE

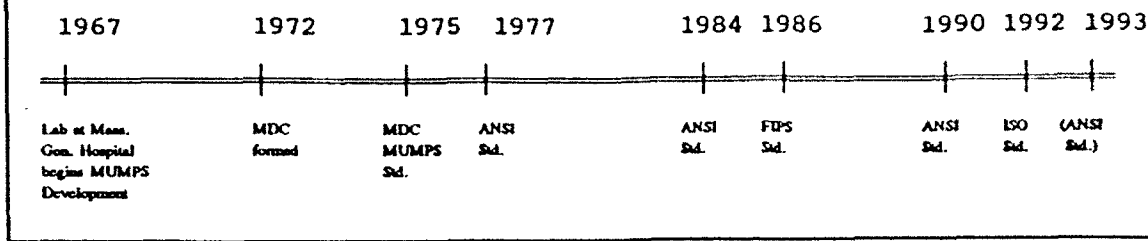
The MDC is composed of subcommittees that address general functional areas of the language, such as user interface or data management. Each subcommittee is further organized into task groups that are responsible for a specific aspect of the language, such as X Windows or string handling. Any organization may request membership in the MDC status; so far, no membership requests have been denied.

Member organization representatives may also join task groups in which they have an interest. The MDC currently includes approximately 10 vendor and 50 user organizations. The member organizations send representatives to the three conferences that the MDC holds each year. Much of the conference time is spent working in task groups. A meeting of the full MDC is held at the conclusion of the conference. The Department of Veterans Affairs (VA) has provided a grant to pay for the annual meeting of the MDC; the DoD and Indian Health Service (IHS) have also periodically provided funds in the past. The MDC is also funded through annual membership dues of \$100.

The MDC has continued to assume the responsibility for maintaining and developing the MUMPS Language Standard. MUMPS has been approved as an ANSI Standard (ANSI/MDC X11.1-1990), a Federal Information Processing Standard (FIPS PUB 125), and an International Organization for Standardization/International Electrotechnical Commission Standard (ISO/IEC 11756). Exhibit 3-5 outlines the various stages of MUMPS development and standardization.

The MDC has published both a MUMPS Language Standard and a MUMPS Portability Standard. The ANSI Language Standard guides implementors on the minimal requirements that must be met in order to conform to the ANSI Standard. The Portability Standard is a subset of the ANSI Language Standard, serving implementors who create MUMPS interpreters and also developers who create MUMPS applications. Each standard is interpreted as a minimum specification to be met by implementors and a maximum specification to be exercised by

EXHIBIT 3-5: TIMELINE OF MUMPS STANDARDIZATION



developers. If both the implementor and developer adhere to the Portability Standard, the portability of the MUMPS program to other platforms will be assured. The existence of the Portability Standard has been a great benefit to MUMPS users. Both MUMPS developers and end users of MUMPS applications are able to take advantage of the higher performance and lower costs associated with newer technology.

The MUMPS Portability Standard means that program developers who adhere to the standard can be confident that their programs will run successfully on any MUMPS implementation. This contrasts with programming languages that lack portability standards. In FORTRAN, for example, a program may comply with the language standard but fail to execute due to the use of arrays that exceed the memory capacity of the target machine.

Implementations of MUMPS may contain extensions to the ANSI Standard. Some implementation-specific extensions support operating system and hardware-dependent utilities. Other extensions to the current ANSI Standard include official "Type A Extensions" that the MDC has approved for inclusion in future standards in the process described below.

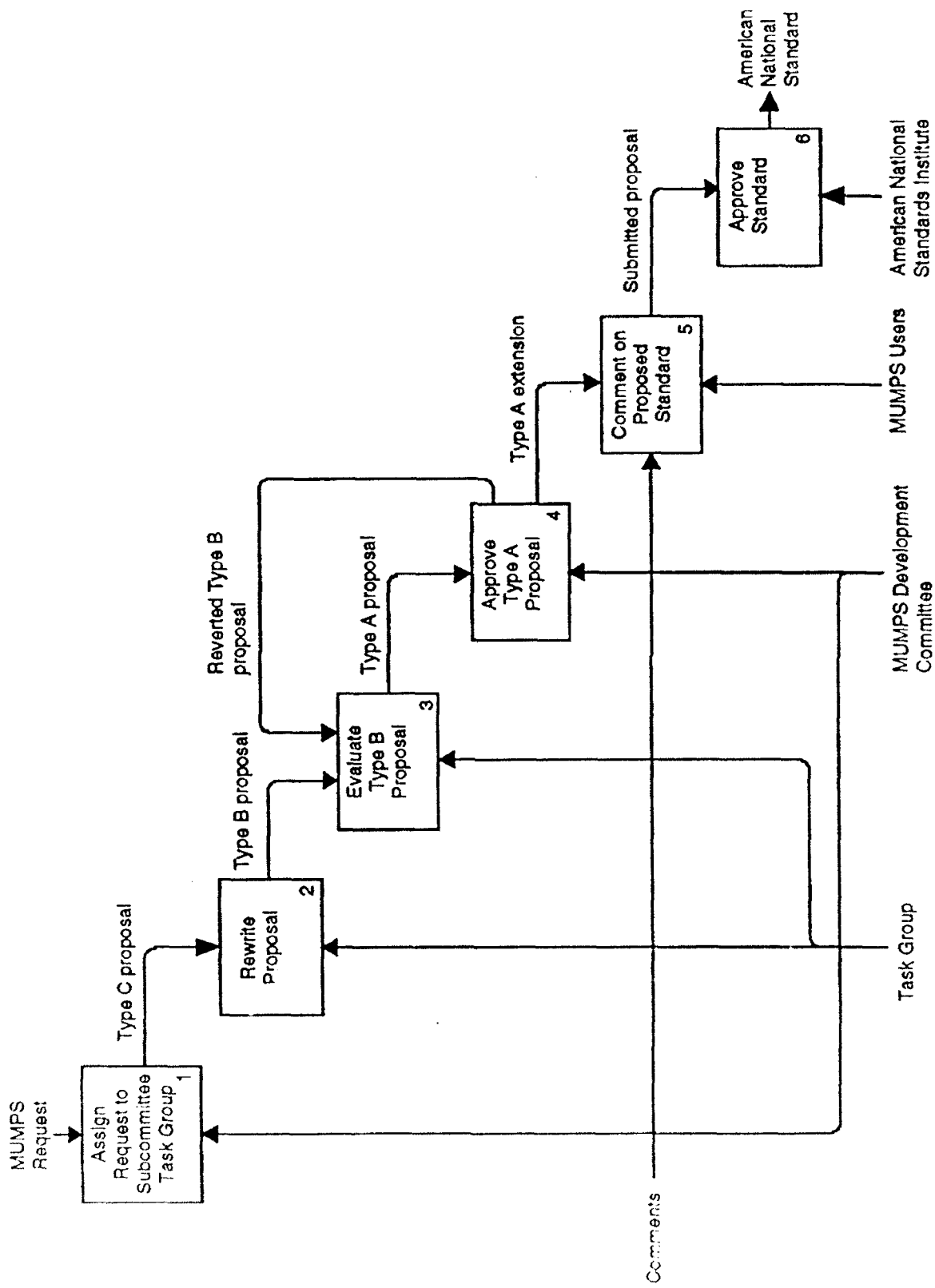
3.2.2 MUMPS STANDARDIZATION PROCESS

Anyone can submit proposals for extensions or modifications to the MUMPS Language Standard. The MDC is responsible for reviewing and approving all such proposed extensions or modifications. The MDC makes the ultimate determination as to whether or not the proposal is adopted as an extension or included in the next submission of the MUMPS Language Standard to ANSI.

3.2.2.1 Standardization by MDC

Exhibit 3-6 shows the MDC's formal process for adopting modifications to the MUMPS Language Standard. The MDC is organized into subcommittees. Each subcommittee is responsible for several task groups. The MDC assigns requests for modifications to the appropriate subcommittee. The subcommittee is responsible for assigning the request to an appropriate task group or forming a new task group if an appropriate one does not already exist.

EXHIBIT 3-6: THE MUMPS STANDARDS PROCESS



A request at this stage is termed a "Type C proposal." The proposal will be rewritten in a formal manner and, if deemed worthy of further consideration, it is termed a "Type B proposal." When the proposal is deemed complete enough to be submitted to the full MDC for consideration, it becomes a "Type A proposal." When a proposal receives type A status, no other changes to the proposal document will be accepted. If it is determined that changes are necessary, the proposal reverts to a Type B proposal and is returned to the subcommittee for modification.

3.2.2.2 Ratification by ANSI

Type A proposals that MDC approves receive "Type A Extension" status. All Type A Extensions are included in the next submission of the MUMPS Language Standard to ANSI. The MDC will seek comments from the MUMPS community on MDC Type A Extensions to the Language Standard. The comments received are submitted to ANSI along with the MDC MUMPS Language Standard. ANSI will approve the MDC MUMPS Language Standard when it is satisfied with the MDC responses to questions on specific proposals that arose out of the canvassing process. The main role of ANSI in the standardization process is to ensure that the MDC has considered the material interests of all affected parties in preparing the MUMPS Language Standard.

The active involvement of the MDC in the continual improvement of the MUMPS Standard has played a significant role in the general portability of MUMPS applications. Proposed extensions to the Language Standard are widely publicized. The MDC actively seeks feedback from MUMPS users on proposed extensions to the ANSI Standard. This collaboration between the standards-setting body and the user community has led to a MUMPS Language Standard to which both MUMPS implementors and application developers closely adhere. Vendor and user consensus over weaknesses in the ANSI Standard are included in future revisions to the standard.

Each subsequent revision to the ANSI Standard has addressed specific vendor and user concerns. Changes to the Language Standard that are not compatible with previous versions are adopted very cautiously. The most significant events and changes in the MUMPS Language Standard include:

- 1977 Standard — Merged the several incompatible dialects of MUMPS that had evolved into a single standard
- 1984 Standard — Implemented the use of descriptive subscripts in MUMPS arrays
- 1990 Standard — Provided support for structured programming techniques
- 1993 Standard (proposed) — Will provide support of MUMPS bindings to industry standards (SQL, X-Windows, GKS)

3.2.2.3 FIPS Adoption by NIST

NIST follows a standard process in recommending to the Secretary of Commerce the establishment or revision of a standard. A sponsoring organization, in this case the MDC, submits a standard to NIST for consideration. NIST elicits comments on the standard from interested or affected parties. Comments are accepted for approximately 3 months. NIST then reviews comments and prepares responses. The sponsoring organization may provide technical background to support NIST's response to the comments. When all comments have been addressed to NIST's satisfaction, the standards will be forwarded to the Secretary of Commerce for approval.

After the approval of the Secretary of Commerce, NIST issues a FIPS PUB specifying the standard. The FIPS PUB becomes effective 6 months after the date of approval. A 12-month implementation period commences on the effective date of the standard. This implementation period gives implementors and developers time to adjust to the adoption of the new standard. At the end of the implementation period, Government usage of the standard becomes mandatory.

As is common, FIPS PUB 125 follows for MUMPS the form of the current ANS. It is not uncommon for a FIPS PUB to be superseded by a recently revised ANS. In this case, the FIPS PUB and the ANS are different until the FIPS PUB is revised.

4.0 MUMPS AND AST

Within the definition of an AST, MUMPS can be evaluated as a technology that attempts to integrate the following environments:

- Programming Language
- Database Management System
- Programming Support Environment (PSE)

A general evaluation of MUMPS features and AST criteria can be performed with respect to these areas. The following sections provide an evaluation of MUMPS with respect to AST criteria considering these technologies. This evaluation considers the language features defined in the MUMPS 1990 ANS and the proposed 1993 ANS. Exhibit 4-1 summarizes the evaluation of MUMPS features against AST criteria.

4.1 MUMPS PROGRAMMING LANGUAGE

The MUMPS programming language provides a great deal of flexibility for a programmer in developing applications. By providing a less constrained development environment, MUMPS assumes a programmer has knowledge about what he or she is attempting to do. Standard programming conventions are largely unenforced within the language.

The following section studies the features of the MUMPS programming environment in relation to AST criteria.

4.1.1 INTERPRETIVE NATURE OF MUMPS

The target platform for MUMPS in its initial design in the late 1960s was an interactive, multi-user minicomputer environment. Storage efficiency to operate effectively in a minicomputer environment was a major concern. The designers of the language also realized that an interactive development environment for the programmers and end users was beneficial in developing functional medical applications. For these reasons it was decided that an interpretive language provided better support for these objectives.

4.1.1.1 Impact on Programmer Productivity and Usability

The interactive programming environment available with interpretive languages such as MUMPS provides good programmer productivity and usability through an interactive development and maintenance environment. Proposed program syntax can be tested immediately since source code is interpreted as statements are entered or executed. Software modifications and corrections can be made and tested immediately. Time-consuming recompilation is not necessary. User-driven development techniques, such as rapid-prototyping, are more easily supported with an interpreted language. This is an advantage where close coordination and

EXHIBIT 4-1: AST CAPABILITIES

| Feature | Productivity | Usability | Maintainability | Portability | Flexibility | Efficiency | Extendibility | Completeness | Maturity | Stability |
|--------------------------|--------------|-----------|-----------------|-------------|-------------|------------|---------------|--------------|----------|-----------|
| Interpreted Language | Good | Good | Good | n/a | Good | Average | n/a | n/a | n/a | n/a |
| Syntax | Good | Average | Poor | n/a | n/a | Good | Fair | n/a | n/a | n/a |
| Data Types | Good | n/a | Fair | Excellent | Excellent | Good | Good | Fair | n/a | n/a |
| String Functions | Good | Excellent | n/a | Excellent | n/a | Good | Average | Excellent | n/a | n/a |
| Math/Stat Functions | Poor | Poor | n/a | Poor | n/a | Poor | Good | Poor | n/a | n/a |
| Control Flow | Average | Fair | Fair | n/a | Average | n/a | n/a | Fair | n/a | n/a |
| Library | Fair | Average | Average | Average | Average | Average | Average | Average | n/a | n/a |
| Data/Record Structure | n/a | n/a | Average | n/a | Average | Good | Good | Good | n/a | n/a |
| Data Retrieval | Fair | Fair | Fair | Excellent | n/a | n/a | Good | Average | n/a | n/a |
| Data Integrity | Fair | n/a | n/a | n/a | n/a | n/a | Good | Fair | n/a | n/a |
| Data Integration | Fair | Fair | n/a | Fair | Fair | n/a | n/a | n/a | n/a | n/a |
| Design Phase | Average | Average | n/a | n/a | n/a | n/a | n/a | Fair | n/a | n/a |
| Edit, Build, Test Cycle | Fair | Fair | n/a | Fair | n/a | Good | n/a | Average | Fair | n/a |
| Configuration Management | n/a | n/a | n/a | n/a | n/a | n/a | n/a | n/a | n/a | n/a |

interaction in the development process between the programmers and end users is required. Programmers are able to demonstrate the effects of changes in the program to end users in an interactive fashion.

4.1.1.2 Impact on Execution Efficiency

Execution efficiency of MUMPS is average. Program code written in an interpreted language will generally execute more slowly than in equivalent compiled code. The interpretation of the code creates additional overhead at execution time. The performance of functions that are CPU intensive will be reduced the most with an interpretive language. Functions that are more input/output (terminal, disk, devices) intensive will be affected less. Most MUMPS implementations tokenize the code to improve execution efficiency.

4.1.1.3 Impact on Maintainability and Flexibility

An interpretive language provides for good maintainability and flexibility. Debugging can be improved with the ability to interact with the program at run time and execute program statements directly from the keyboard. The location of errors and state of variables during execution are immediately available.

However, to reduce the overhead of interpreting MUMPS program code, abbreviated command syntax is often employed to reduce the number of characters interpreted at execution time. The abbreviated command syntax impairs the readability and understandability of program code. Strong controls over coding style and conventions are necessary to prevent the use of syntax or styles that will harm the maintainability of source code. Tokenizing the code prior to execution allows the use of more complete and meaningful syntax, improving maintainability without degrading execution efficiency.

4.1.2 *MUMPS SYNTAX*

The command syntax employed by MUMPS is terse and abbreviated, partly to provide good execution efficiency. Significant use of symbolic characters is also used in standard MUMPS syntax. Many existing MUMPS applications use the abbreviated form of MUMPS commands and include numerous commands on the same program line. Productivity is good with MUMPS programming syntax. Usability is average. Although the language syntax is not as natural as other languages, there is a limited set of commands to be learned. New programmers can become quickly proficient in the language. Maintainability and extendability as a result of the syntax are fair. The syntax of MUMPS can harm maintainability due to poor readability of program code. The readability and maintainability of program code are improved by enforcing programming conventions on proper coding style.

4.1.3 DATA TYPES

The fact that MUMPS data is physically stored as character strings has a strong impact on the flexibility and efficiency of the language in processing data. All data, whether numeric or character, is physically stored as character strings. Data type declarations are not supported in MUMPS. Since data is physically stored as character strings, MUMPS employs type coercion, where data is automatically converted to a type based on the operation being performed on the data. A data item with many different operations being performed on it during the execution of a program may assume many different types. MUMPS data may be interpreted as numeric, character string, or boolean true-false values, providing a fair level of completeness. The operations on data are not constrained to a limited set as with type checking.

With type coercion, MUMPS provides excellent flexibility and power to the programmer. The extendability of the language in this area is good, as shown by the VA's Fileman, for example. The programming constraints enforced with type checking are eliminated in MUMPS. Maintainability is improved where simple changes in the definition of data are necessary. Such a change requires little or no modification where a similar change in a strongly typed and declared language could result in significant modifications. However, this additional flexibility and power is hazardous when misused. A MUMPS programmer *must* understand what he or she is attempting to do. Overall, fair maintainability is achieved with data typing in MUMPS. Data typing in other languages eliminates execution errors by catching these errors during the compilation process.

4.1.4 STRING MANIPULATION OPERATORS AND FUNCTIONS

MUMPS contains many operators and intrinsic functions to manipulate character strings. The string manipulation abilities of MUMPS are among the primary strengths of the language. They are derived from the initial objective of the language to support text-based applications. MUMPS has excellent completeness and usability in string operators and functions, providing functions for character and string conversion, parsing, replacement, pattern matching, and concatenation of characters and entire character strings. The extendability of these functions is average. The string operators and functions provide good productivity and efficiency for programs working with and manipulating textual data.

4.1.5 MATHEMATICAL/STATISTICAL OPERATORS AND FUNCTIONS

One of the largest drawbacks to the MUMPS language is its poor completeness for complex mathematical and statistical functions. MUMPS does provide a standard set of numeric operators (addition, subtraction, multiplication, division, integer division, and remainder). An exponentiation operator will be included in the 1993 ANS.

MUMPS currently supports through type coercion a numeric (floating-point) and integer type. Mathematical precision is limited to 12 significant digits. MUMPS implementations supporting

more than 12 significant digits of precision are hardware dependent. Therefore, the reliability of MUMPS applications requiring more than 12 significant digits of accuracy cannot be assured.

4.1.5.1 Impact on Execution Efficiency

Since data is physically stored as character strings, execution efficiency is poor with mathematical and statistical functions. MUMPS has an inherent performance limitation in performing complex mathematical and statistical functions. When performing mathematical functions, MUMPS converts a character string (physical storage format) to a numeric representation (binary format), performs the mathematical functions on the binary data, and converts the result to a character string for storage. The interpretive nature of the language is a hindrance in the performance of these functions, which are more CPU intensive.

4.1.5.2 Impact on Productivity, Portability, and Extendability

Since MUMPS is lacking in standard (intrinsic) libraries of mathematical and statistical functions, it provides poor productivity for applications whose primary activities are complex mathematical or statistical functions. However, the extendability of the language is good through support for external routines (C, FORTRAN, etc.) that perform these functions. This feature has been provided by MUMPS vendors and will be standardized by the MDC in the 1993 ANS.

Vendor implementations provide some mathematical and statistical functions. However, since these may be implementation specific, portability is poor.

4.1.6 PROGRAM CONTROL STRUCTURES

MUMPS provides a smaller set of program flow control structures than other languages. MUMPS control structures include IF, ELSE, FOR, an argumentless DO, and GOTO commands. Currently, MUMPS does not provide CASE or WHILE control structures, although these can be simulated with the existing commands. The manner in which these control structures are implemented in MUMPS differs from other languages. The statements under the scope or control of the IF, ELSE, and FOR commands must be on the same line as the command.

4.1.6.1 IF/ELSE Conditional Flow Control

The IF and ELSE commands are syntactically separated. The IF statement sets the value of a special variable (\$TEST) to either true or false. If \$TEST is true, the remainder of the commands on the "IF line" are executed. These commands could be a DO command to execute a nested block of statements or to transfer control to a subroutine. If \$TEST is false, execution immediately terminates and resumes on the next line of the program.

Unlike most programming languages, the ELSE command in MUMPS is not syntactically linked to a previous IF command. The ELSE command merely checks the value of \$TEST; and if it

is false, MUMPS executes the remainder of commands on the ELSE line. An ELSE is necessary when alternative statements are to be executed when an IF statement evaluates to false.

The manner in which the IF/ELSE control structures are implemented creates a hazard for the MUMPS programmer. Like other MUMPS variables, \$TEST is implicitly available to all routines. Therefore, when invoking additional IF/ELSE commands in a nested block or subroutine, programmers must be careful that the value of \$TEST is correct when control returns to the main routine. It is possible that \$TEST could be set to false by an additional IF command. When control returns to the main program, the ELSE commands would erroneously execute. Exhibit 4-2 illustrates how this may occur in a MUMPS program.

EXHIBIT 4-2: EXAMPLE OF IF/ELSE FLOW CONTROL

The following is an abbreviated example of a MUMPS program using the IF/ELSE program flow structures (text following a semicolon ";" is a comment):

```
;Illustrate use of IF/ELSE and $TEST variable
Read !, "Enter Date of Birth",DOB
If DOB{"1971" DO ELIG(parameter list)
Else Do
.      (nested statements)
.      (nested statements)
Quit

; -----
Sub-Routine ELIG

ELIG(parameter list)
;Procedure ELIG. Unintentional reset of $TEST
Read !, "Are you a full-time student?",ANS
If ANS{"Y" Do
.      (nested statements)
.      (nested statements)
Else
Write #!! "You are not eligible"

; -----
```

The \$TEST variable is reset at every IF command and checked at every ELSE. If the DOB contains 1971, \$TEST is set to true and the sub-routine ELIG is invoked. If the ANS is "N" in the ELIG routine, \$TEST is set to false and control returns to the main routine. Since \$TEST is false upon the return to the main routine, the ELSE statements will be executed, even though the previous IF had previously set \$TEST to true.

4.1.6.2 Block Structuring With DO

An argumentless form of the DO command was recently added to the language standard to improve program flow control by minimizing the need for GOTOs. This command provides a mechanism to write more structured and readable MUMPS code. A nested block of statements following the argumentless DO command will be executed. When used properly, block structuring with the DO command will aid in the readability and maintainability of MUMPS programs. Exhibit 4-3 provides an example of the use of the DO command with block structuring and parameter passing.

EXHIBIT 4-3: EXAMPLE OF BLOCK STRUCTURING AND PARAMETER PASSING

The following MUMPS program employs block structuring with the DO command to conditionally execute statements in a program:

```
PTN      ;Records patient allergy and ailment information prior to
        ;the performance of any services in a clinic or hospital.
        Read !,"Is the patient allergic to any medication (Y/N)?"
        If ANS["Y" Do
        .   For N=1:1 Read !,"List medication:",MEDIC
        .   ...Quit:MEDIC=""
        .   ...Read !,"Describe reaction:",REACT
        For N=1:1 Read !,"Describe ailments:",AIL Quit:AIL=""
        ...Read !,"Enter date of ailment:",DATEAIL
        Quit
```

The following MUMPS program illustrates parameter passing with the DO command. The subroutine PVALUE calculates the present value of a lump sum payment for 1 interest period.

```
PVALUE(I,P)
        ;Calculate present value of lump sum. I=interest ;P=Principal. Assumes that calculation
        ;is for one interest period.
        Set PV=P/(1+I)
        Write !,"Present value is: ",PV
```

Calling the program PVALUE with the Do statement...

```
Do PVALUE(.08,1000)
```

...yields the following results

```
Present value is: $925.93
```

The values .08,1000 in the actual list are passed by position to the variables I and P in the formal list. An explicit New function is performed on the variables in the formal list (I,P).

4.1.6.3 FOR Loop Flow Control

The FOR command provides a mechanism to execute a series of statements (loop) repeatedly based upon a conditional value. The FOR command can be used to increment a value (counter), repeating the following sequence of statements until a specific value is reached. The statements to be conditionally executed must reside on the same line as the FOR command. A DO or Xecute command appearing at the end of the line can be used to execute a block of statements or invoke a subroutine.

4.1.6.4 Transfer of Control With DO and Xecute

The DO and Xecute commands are used to invoke a subroutine. A DO command with arguments (routine name, parameters) will execute a subroutine. An Xecute command will execute a string of commands stored in a variable. MUMPS has this capability due to its interpretive nature. Control will return to the main program upon termination of the subroutine. The subroutines may return values back to the main routine.

4.1.6.5 Impact of Control Structures on AST Criteria

The MUMPS control structures generally result in less structured MUMPS program code. The potential problem with the IF/ELSE commands and less structured code can affect maintainability. The completeness and maintainability of the control structures are fair. The control structures combined with standardized support for structured programming techniques provide average productivity, usability, and flexibility.

4.1.7 PROGRAM ROUTINE LIBRARY SUPPORT

The addition of parameter passing in the 1990 ANS allowed MUMPS programmers to develop libraries of reusable software routines. This capability has long been found in most languages. MUMPS, however, lacks the support for strong data typing and inheritance necessary to construct programs from generic, reusable routines. These capabilities are found in languages that directly support object-oriented programming techniques. Portability is average since library routines have not been standardized. Currently, MUMPS provides fair productivity, average maintainability, and average usability for program routine libraries.

4.1.8 INTRAPROCESS COMMUNICATIONS

The global scoping of variables in MUMPS complicates intraprocess communication. Variables are defined at the time of their use and are implicitly available to all routines using the same working area. Although this may provide good productivity and flexibility, maintainability is poor as a result of the inability of MUMPS to implicitly hide locally used data from other routines.

Parameter passing and other commands are available to control the scope of variables, reducing the risk of unintentionally re-using variables. Parameter passing was added to the 1990 ANS to improve communication between routines within a process. When variables are passed as parameters, information hiding is accomplished by the resetting of local variables to null values.

Most programming languages provide for the explicit definition of variables in a separate data declaration section of a program. MUMPS does not provide for data declarations. Data is declared at the time of its use in the program. The lack of explicit data declarations provides good flexibility. However, maintainability is poor since the understandability of a program is reduced and the identification of errors associated with variable names will be more difficult to isolate.

4.2 DATABASE MANAGEMENT SYSTEM

In addition to providing an interactive programming environment, one of the principal objectives of the original design of MUMPS was to provide an integrated, shared database system within a programming language. This feature of MUMPS is probably its most distinguishing characteristic in comparison with the more popular programming languages. Several factors guided the design of the MUMPS database structure:

- Efficient storage of unstructured, textual data
- Ability to operate with minimal hardware requirements
- Requirement for easily shared data.

The following sections compare the data management features of the MUMPS language with those commonly found in modern database management systems.

4.2.1 DATA STRUCTURES AND STORAGE

A MUMPS database supports variable-length strings within a sparse, dynamic hierarchical array structure. MUMPS internally handles storage allocation for an array. Programmers do not have to anticipate the dimensions of an array when defined in a MUMPS program. When MUMPS was conceived in the late 1960s, the relational model was years from becoming a reality and the hierarchical model was determined to be the most efficient method of handling data in a medical environment. Today, tools have been developed by vendors and others, such as InterSystems M/SQL and the VA's Fileman program, that provide relational capabilities to a MUMPS database. Standardization of embedded SQL commands in a MUMPS program will be included in the 1993 ANS.

Today, most database systems retrieve and update information in units termed records. In this form, a record usually contains several fields that represent attributes that are dependent upon the instance of that record. MUMPS retrieves and updates information according to a node. A node may contain a single value, similar to a field. However, to replicate the concept of a

record and logically group data as a single unit, several items may be stored in a node. Each item (field) in the node will be delimited by a comma or some other unique character, forming a variable length record.

MUMPS provides excellent storage efficiency for textual data. Trailing blanks or null values in data do not consume storage space. Neither do they consume input-output time, thus improving execution efficiency. However, the storage of numeric data with many significant digits is much less efficient when stored in character string format than could be obtained in binary format. Hierarchical database systems generally offer good execution efficiency when compared with relational DBMSs. Relational set operations requiring multi-table joins create significantly more processing overhead.

The dynamic array structure and variable length records in MUMPS provides for average maintainability and flexibility. A MUMPS program is adaptable to unanticipated changes in the number and size of records and fields. However, maintainability and flexibility are hampered since a hierarchical array structure does not support multiple user views of data commonly found in relational database systems. Redundant data may be necessary to support multiple user views if the data does not map well into the hierarchical model. This redundant data will reduce storage efficiency.

4.2.2 DATA RETRIEVAL

In order to retrieve a node the structure of the global must be known in advance. Compared to hierarchical structures, relational database systems can simplify the retrieval of data. To counter the complexity of traversing a MUMPS global to retrieve data, intrinsic functions are built into the language to simplify the code required to retrieve data. Tools are available to facilitate data retrieval from a MUMPS global.

MUMPS data retrieval provides fair productivity, usability, and maintainability. MUMPS operates on data one record at a time instead of the set orientation of a relational DBMS. More programming effort is required to retrieve data than in a relational database system. Embedded SQL syntax in a MUMPS program will be standardized in the 1993 ANS, providing good extendability of retrieval functions. The support for logical user views of data is also a significant advantage in data retrieval from a relational DBMS. A MUMPS hierarchical database does not directly support logical user views of data. Program maintenance is impaired when additional user views are required. Often user requirements for multiple views cannot be anticipated in advance.

4.2.3 DATA INTEGRITY/SECURITY

MUMPS has features that assist in maintaining the integrity of the database during user updates and retrieval activities. Unlike many modern DBMSs, however, these features are not implicitly handled by the DBMS and must be explicitly defined in the MUMPS program.

Because data integrity and security functions must be explicitly coded in a MUMPS program, the productivity and completeness of the language is fair for integrity and security issues when compared to modern DBMSs.

4.2.3.1 Data Locking

Without proper synchronization between the updating and access to data in a multi-user environment, the chance of lost data exists. MUMPS supports the locking of data to synchronize the access and updating of data. A locking mechanism can be employed to prohibit updates to an entire global or specific nodes within the global until the lock is removed. The ability to lock data on a global, branch, and node level is an important feature for multi-user database applications.

Most modern DBMSs perform this lock implicitly. With in MUMPS, an explicit lock command must be coded to lock data. The program code must set and check for the existence of lock flags when retrieving and updating data. By default, the MUMPS lock command automatically releases previous locks set by the user. This eliminates the risk of a deadlock situation, where two users simultaneously lock data items each is attempting to access.

4.2.3.2 Transaction Processing

Transaction processing is an extension to record locking, ensuring integrity of all data involved in a transaction. Transaction processing is not currently supported in the ANS language standard. Additional commands are proposed in the 1993 ANS that will provide support for transaction processing. Some vendor implementations of MUMPS have included support for error processing in their products. Transaction processing support is an important feature in maintaining database integrity in transaction-oriented applications and is a feature of many modern DBMSs.

4.2.3.4 Security

System security in the form of user sign-on security and restricted access to programs and data is not directly handled within MUMPS. Many modern DBMSs provide these features to control user access and updates to data.

4.2.4 DATA INTEGRATION

Data between MUMPS systems can be integrated and accessed across platforms. Like many DBMSs, data can be dispersed across several locations and shared across applications. In MUMPS, references to the remote environment must be indicated in the subscript to the variable.

It is possible to import and export data to and from a MUMPS global. The MUMPS language standard does not, however, define standard routines to perform import/export functions. Routines for this function are usually added by implementors, but they may not be portable

across MUMPS implementations. Modern DBMSs have functions to import and export data. If the implementor provides no standard routines, routines can be built within MUMPS to provide this capability. The string manipulation functions of the language provide a great deal of flexibility in the ability to parse data into the desired export format. This flexibility is offset by the fact that only character data can be read by a MUMPS program. Importing/exporting of binary data is not supported.

Physical file storage is not standardized in MUMPS and differs between vendor platforms. This lack of standardized file handling is currently a drawback with MUMPS. Data within a global can generally only be accessed by a MUMPS routine. Typically, the operating system recognizes a global as a file, but is unable to directly access the contents of the global. As a result, operating system utility functions, such as archiving and restoring workspaces, are frequently unable to operate on a MUMPS database.

Portability with data integration is fair due to the lack of standardized import/export functions and file handling. Usability likewise will be fair. Considerable programming effort may be required to import and export data if vendor routines cannot be used. Flexibility is fair considering that MUMPS has limitations in dealing with binary data, although its ability to parse character data is excellent.

4.3 PROGRAMMING SUPPORT ENVIRONMENT

Software products are available that provide support for editing, saving, loading, debugging, and testing programs. Other products such as design aids, code generators, source code version control aids, etc., are available. The products available encompass the Programming Support Environment for MUMPS.

4.3.1 SPECIFICATION/DESIGN PHASE

MUMPS as an interpreted language provides a rapid prototyping environment to support the specification/design phase. A prototyping environment provides advantages where user feedback and interaction is required to further define program requirements. However, most CASE tools to support program design and specification do not support the generation of MUMPS program code. The completeness of MUMPS in this area is fair. Fewer CASE tools directly support MUMPS than is the CASE with other languages. Overall, MUMPS provides average productivity and usability in the specification/design phase.

4.3.2 EDIT/LINK/TEST CYCLE

MUMPS implementors provide utilities to support program editing, loading, saving, and debugging. These products are not standardized across MUMPS implementations, but many could be used independent of the language implementation. The interpretive nature of the language provides an interactive debugging and testing environment, resulting in good efficiency.

However, fewer tools are available for MUMPS than for many other languages. Overall, MUMPS offers fair productivity, portability, and usability.

4.3.3 *CONFIGURATION MANAGEMENT*

Configuration management provides an automated mechanism to manage multiple versions of source code. This is particularly useful for large projects in coordinating and managing source code development and revisions. Unlike some languages, MUMPS does not provide standard mechanisms such as routine libraries to support configuration management.

5.0 SUMMARY

This report analyzes the MUMPS programming language and environment with respect to AST categories and criteria. The information in this report was gathered through interviews with staff within NIST, to determine whether an evaluation of MUMPS had already been conducted, and what the result was. Contacts provided by MUG and MDC members were consulted to gather information about the MUMPS programming language environment. Contacts at Robert Morris College were interviewed to gather additional information about the MUMPS programming language and environment.

MUMPS is unique, in that it combines programming language services, database management services, and programming support services. More commonly, these services are provided through separate languages, systems, and tools. The MUMPS programming language and environment provide significant improvements over many of the capabilities currently in use. However, there are other AST criteria in which MUMPS lags other, more developed programming languages and tools.

5.1 AST CATEGORY OF MUMPS

The AST definition enumerates the types of technologies by which the MUMPS programming language and environment may be categorized:

- Software Tool
- Life Cycle Support Environment
- Programming Support Environment
- Nonprocedural Language
- Database Management System
- Other Technology

5.1.1 SOFTWARE TOOL

As previously defined, a software tool is a computer program used in the development, testing, analysis, or maintenance of a program or its documentation.

MUMPS is a programming language, and could be used to make software tools. However, MUMPS itself is not a software tool or a computer program. A MUMPS interpreter qualifies as a software tool; however, this study documents the specifications of MUMPS with respect to AST, not implementations.

5.1.2 LIFE CYCLE SUPPORT ENVIRONMENT

As defined in chapter 2.0, a life cycle support environment is an integrated set of software tools and utilities that are designed to support a software product from the time it is conceived until it is no longer used.

The MUMPS programming language standards do not incorporate life cycle support in their definition. Neither do the programming language standards for many other languages such as Ada, C, FORTRAN, and Pascal. Commercial and proprietary life cycle support environments have been developed for MUMPS and other languages. A standard life cycle support environment targeted toward Ada is under proposal to the DoD; the DoD is expected to award a contract in 1993.

5.1.3 PROGRAMMING SUPPORT ENVIRONMENT

As defined in chapter 2.0, a Programming Support Environment is an integrated collection of software tools accessed via a single command language to provide programming support capabilities throughout the software life cycle.

Most other standard programming languages, such as C, FORTRAN, and Pascal, do not incorporate PSEs in their definitions, although commercial and proprietary PSEs have been developed for them.

The DoD has developed a standard (MIL-STD-1838A) for a common Ada PSE (APSE) interface; however, the interface standard is still separate from the language standard. The Common APSE Interface Set - revision A (CAIS A) also has a validation test suite to assess conformance with the standard.

Through implementation-specific commands, MUMPS interpreters provide program loading, editing, and saving capabilities as in a PSE. Although these commands are not standardized, MUMPS contains a rudimentary PSE. More complete commercial and proprietary PSEs have been developed for MUMPS but have not been standardized.

5.1.4 NONPROCEDURAL LANGUAGE

From the previous definition, a nonprocedural language is one in which the user states what is to be achieved without having to state specific instructions that the computer must execute in a given sequence.

Programming languages, such as Ada and C, are procedural by definition, as the programmer explicitly sequences the commands that the computer must execute. Some database languages, such as SQL, are nonprocedural since the user specifies only the desired result, and the database manager will automatically determine how to execute the command.

MUMPS has aspects of both procedural languages and nonprocedural languages. MUMPS uses routines to structure and sequence the instructions commands that the computer will execute. The MUMPS programming language is clearly a procedural language. However, the intrinsic MUMPS database management commands are nonprocedural. For example, when the user directs MUMPS to store data, MUMPS will automatically allocate sufficient storage space, assign the data values, and update indices to the new data. The MUMPS nonprocedural database

management commands are typically embedded within the procedural programming language for sequencing and structure. This is akin to embedding SQL statements within another programming language.

5.1.5 MODERN DATABASE MANAGEMENT SYSTEM

As previously defined, a DBMS is a computer-based system used to establish, make available, and maintain the integrity of a database. A DBMS may be invoked by nonprogrammers or by application programs to define, create, revise, retire, interrogate, and process transactions; and to update, back up, recover, validate, secure, and monitor the database.

MUMPS directly supports persistent data that is shared among users of the MUMPS system. MUMPS currently provides a simple record locking mechanism; full transaction processing has been proposed for the 1993 ANS. MUMPS has integrated database management capabilities.

5.1.6 OTHER TECHNOLOGY

The "other technology" clause in the definition of AST includes all other technologies not covered among the technologies explicitly enumerated.

MUMPS incorporates aspects of many of the above categories with a programming language. MUMPS should also be considered a technology that integrates the AST technologies of a programming language, database management system, and rudimentary programming support environment.

5.2 AST CAPABILITIES OF MUMPS

The MUMPS programming language and environment can be evaluated with respect to the following AST capabilities:

- Productivity
- Usability
- Maintainability
- Portability
- Flexibility
- Efficiency
- Extendability
- Completeness
- Maturity
- Stability

Because MUMPS integrates DBMS services with a programming language and a rudimentary PSE, each aspect of MUMPS should be considered with respect to the above AST capabilities.

Since MUMPS is a composite technology in which the component technologies cannot be used separately, a composite rating must also be established. To evaluate MUMPS against each of the above AST capabilities, the following qualitative scale was chosen:

- Excellent
- Good
- Average
- Fair
- Poor

A rating of "Excellent" means that MUMPS is clearly superior to the typical system commonly in use. Conversely, a rating of "Poor" means that MUMPS is clearly inferior to the typical system commonly in use. Exhibit 5-1 uses this scale to summarize the ratings of MUMPS against the AST criteria. In some cases, MUMPS provides superior capabilities to typical systems; however, in other cases, MUMPS is less capable than typical technologies.

EXHIBIT 5-1: MUMPS AST CAPABILITY ASSESSMENTS

| <u>AST CRITERION</u> | <u>1990 ANS</u> | <u>1993 ANS</u> |
|----------------------|-----------------|-----------------|
| Productivity | Good | Good |
| Usability | Average | Good |
| Maintainability | Average | Average |
| Portability | Excellent | Excellent |
| Flexibility | Average | Average |
| Efficiency | Good | Good |
| Extendability | Average | Average |
| Completeness | Average | Good |
| Maturity | Average | Average |
| Stability | Fair | Fair |

5.2.1 PRODUCTIVITY

For a programming language, productivity is the ease with which systems and programs, and their component functions and procedures can be developed and written using MUMPS. As applied to a DBMS or PSE, productivity is the extent to which the user can develop more application programs in a shorter amount of time.

MUMPS has good productivity as compared to other programming languages. The original language design emphasized writeability; however, some of the aspects of the language that promote productivity adversely affect subsequent maintainability. For example, the abbreviated syntax allows users to write code quickly because they do not have to type as many characters. The lack of declarative statements causes similar results. Because MUMPS is interpreted, users can develop and test programs and systems quickly because they do not need to wait for the code to be compiled. The high productivity of MUMPS allows programmers to use rapid prototyping when developing systems. However, the decreased number of structured programming constructs forces programmers to use less obvious methods in many cases.

The integral DBMS assists the user in developing systems and programs by removing the overhead normally required to integrate and synchronize the individual systems. However, as a stand-alone DBMS, MUMPS has fair productivity. Typical DBMSs do a better job at data retrieval, since they are set oriented as opposed to record oriented. When more than one record has to be operated upon, the programmer will have to direct the system to operate on the other records in the set. The planned SQL binding for MUMPS will allow the programmer to perform set-level commands.

5.2.2 USABILITY

As applied to a programming language, usability is the ease with which a user can learn the programming language, write correct programs in the language, and understand programs in the language. For a DBMS, usability is the ease with which a nonprogrammer can manage, enter, and extract data from the system. With respect to a PSE, usability is the extent to which the user is assisted in developing programs.

The overall usability of the MUMPS programming language is good. Because MUMPS requires only limited knowledge to write programs, it is easy for users to learn. The ability to develop and test programs quickly aids the user in writing correct programs in the language. The optionally abbreviated syntax of the language, combined with the lack of declarative statements, works against the ability of new users to understand MUMPS programs. Nonetheless, users claim that MUMPS has a short and shallow learning curve.

5.2.3 MAINTAINABILITY

With respect to the MUMPS programming language, maintainability is the ease with which a program written in MUMPS can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment. Similarly, for a DBMS, maintainability is the ease with which data stored the system can be modified or stored differently.

Overall maintainability of MUMPS is average, because it is easier to determine the cause of a fault in an interpreted system. The four thousand character limit on code size enforces modularity of MUMPS code, which improves maintainability. The optional block structuring of MUMPS code also improves its maintainability; users report that block structuring is typically mandated in their code standards for development. As in any programming language, good program design will greatly improve maintainability of the resultant code. Maintainability is especially hampered by the optionally abbreviated syntax of MUMPS programs and lack of declarative statements. Because MUMPS routines tend to be small due to the enforced modularity and terseness of the language, users often choose to rewrite code, rather than to modify the existing code.

5.2.4 PORTABILITY

For a programming language such as MUMPS, portability is the ease with which a computer program written in MUMPS can be transferred from one hardware environment, software environment, or vendor implementation to another. Similarly, for a DBMS, portability is the ease with which the DBMS and the data stored within it can be transferred between environments.

Traditionally, MUMPS has had excellent portability between and among hardware environments, operating systems, and vendor implementations. MUMPS is the only standardized programming language that also defines a Portability Standard. The MUMPS Portability Standard covers implementation-dependent issues, such as the minimum number of significant figures for numbers or the maximum string size. The Portability Standard guarantees functional portability, whereas the Language Standard guarantees syntactic portability.

MUMPS portability is roughly equivalent to Ada portability. By DoD mandate, Ada supersedes and subsets may not use the "Ada" name. This approach results in high syntactic portability. Functional portability, however, is not guaranteed for Ada programs. A program developed on a large machine may allocate more space, or require more numerical precision, than is available on a smaller machine. In such a case, the program could not be compiled or run on the smaller machine.

5.2.5 FLEXIBILITY

As applied to MUMPS, flexibility is the ease with which MUMPS programs can be modified for use in applications or environments other than those for which they were specifically

designed. For a DBMS, flexibility is the ease with which the DBMS and its data can be modified for use in new applications.

MUMPS has average flexibility due to enforced modularity, ease of developing programs, and system independence. However, as with any programming language, MUMPS programs must be designed to be modifiable in the future.

5.2.6 EFFICIENCY

With respect to MUMPS, efficiency is the degree to which MUMPS programs perform their functions with a minimal consumption of storage space and time.

Traditionally, MUMPS has had excellent storage efficiency, and good execution efficiency. Because MUMPS does not store or allocate either trailing blanks in strings or null values, text data is stored and manipulated very efficiently. However, because all data in MUMPS is stored as strings as opposed to the internal machine representation, numerical data with many significant digits will be stored less efficiently. Because MUMPS implementations have excellent storage efficiency and allocate minimal resources to users, execution efficiency can be good if user processes are highly interactive; MUMPS will not have to move as much data to accomplish a given task. If intensive mathematical calculations are to be performed, the relative lack of support for efficient numerical representations and calculations will result in poor execution efficiency.

5.2.7 EXTENDABILITY

For a programming language such as MUMPS, extendability is the ease with which the language's functional capabilities can be increased.

Overall, MUMPS has average extendability. MUMPS currently supports extrinsic function calls. These extrinsic function calls are supplied by the vendor or written by the developers. In addition, vendor-specific extensions currently allow users to access program libraries; these extensions are in the process of being standardized. As with most standard programming languages, the libraries are implementation dependent. Only C has defined a standard library that all implementations can be expected to support.

5.2.8 COMPLETENESS

As applied to a programming language such as MUMPS, completeness is the degree to which the standard provides definitions for key features that support programming services. For DBMSs, completeness is the degree to which the system supports data management and transaction processing. With respect to a PSE, completeness is the degree to which the PSE assists the programmer in the steps of developing applications in a shorter amount of time.

Traditionally, MUMPS has had excellent string and text handling but poor capabilities for

mathematical and statistical calculations. Bindings to MUMPS add graphics and networking. The extrinsic function capabilities for handling advanced mathematical calculations and the external library support allow vendors to address areas where MUMPS is less complete.

Currently, MUMPS is less complete than are typical DBMSs. For example, transaction processing capabilities are being added to MUMPS that most other DBMSs already have.

5.2.9 MATURITY

As defined in chapter 2.0, maturity is the degree to which the specification and its underlying technologies concepts are well understood. This definition applies to programming languages, DBMSs, and PSEs.

Overall, MUMPS is a mature technology. Precursors to MUMPS were developed 25 years ago. MUMPS has undergone three ANS standardizations since its inception in 1967. A fourth ANS standardization is anticipated to be completed in 1993, and a fifth standardization is expected for 1996.

The typical relational DBMS (RDBMS) is based on a mathematical model of set theory. This mathematical model displays greater maturity than MUMPS. PSEs are still relatively new, and are still being developed; however, PSEs exist that provide directly to the programmer than does greater functionality to the programmer directly MUMPS.

5.2.10 STABILITY

As defined in chapter 2.0, stability is the likelihood that the specification will on have many or significant changes within the next 2 years. This definition is also applicable to programming languages, DBMSs, and PSEs.

The planned revision to the MUMPS Language Standard reveals that MUMPS is not stable in the near term, since major changes are planned within 2 next years. However, the changes that are anticipated will help reduce the number of changes to the MUMPS programming language for the long term. The standardized support for external libraries and procedures allows many such changes to be made through standardizing a new library, rather than modifying the MUMPS language itself. MUMPS will be able to integrate with industry and Open Systems Environment standards.

The MUMPS Language Standard continues to evolve to meet changing user needs, with revisions to the standard planned every 3 years. This is in sharp contrast to most other standards which are revised on a 6 to 10 year basis. For example, the Ada and SQL standards, are being revised about 10 years after their initial standardization. Similarly, the FORTRAN standard has been revised about every 10 years, while the COBOL standard has been revised about every 6 years.

This apparent instability is largely offset by the existence of a Portability Standard, a strong history of backwards compatibility, and the fact that most changes are driven by the users of the language. In addition, the standards process allows proposed changes to be evaluated before they are incorporated into the language. This helps to minimize the likelihood of incompatible vendor-specific extensions. On net, MUMPS has average stability; without the Portability Standard, MUMPS has fair to poor stability.

5.3 CONCLUSIONS

The MUMPS programming language and environment are becoming more open and more general purpose. However, the original design of the language and the importance of backwards compatibility continues to exert a strong influence on current and future versions of MUMPS. MUMPS remains very well suited toward handling shared variable-length text-based data.

MUMPS provides significant advantages in the areas of productivity, usability, and portability over other programming languages. MUMPS has equivalent extendability, efficiency, and maintainability in comparison to other languages. There are other programming languages that provide superior completeness, maturity, and stability over MUMPS.

MUMPS provides greater productivity, usability, and efficiency as compared to other modern database management systems. Modern database management systems display greater maturity and completeness over MUMPS.

MUMPS is generally inferior to typical PSEs. MUMPS provides minimal support for rapid prototyping through program loading, editing, saving, and debugging. MUMPS lacks many other features that typical PSEs have, such as program design support, automated testing support, and configuration management support.

Although MUMPS can be considered as a programming language, DBMS, or PSE separately, it is more useful to consider MUMPS as a unified whole. The additional overhead required to integrate the system components must be weighed against the particular disadvantages that MUMPS has with respect to the particular component. For small to moderate-sized applications that primarily focus on shared text-based data, MUMPS would be an excellent choice. As the application grows larger, or becomes more dependent upon mathematical calculations, MUMPS becomes less attractive.

APPENDIX

APPENDIX A.1: ABBREVIATIONS

| Acronym | Definition |
|----------|---|
| ANS | American National Standard |
| ANSI | American National Standards Institute |
| APSE | Ada Programming Support Environment |
| AST | Advanced Software Technology |
| CAIS A | Common APSE Interface Set Revision - A |
| CIVC | CAIS A Implementation Validation Capability |
| DBMS | Database Management System |
| DoD | Department of Defense |
| DoDD | Department of Defense Directive |
| FIPS | Federal Information Processing Standard |
| FIPS PUB | Federal Information Processing Standard Publication |
| GKS | Graphical Kernel Services |
| GUI | Graphical User Interface |
| HOL | Higher Order Language |
| IEC | International Electrotechnical Committee |
| IEEE | Institute of Electrical and Electronics Engineers |
| IHS | Indian Health Service |
| ISO | International Organization for Standardization |
| MDC | MUMPS Development Committee |

| | |
|-------|---|
| MUG | MUMPS Users Group |
| MUMPS | Massachusetts General Hospital Utility Multi-Programming System |
| NIST | National Institute of Standards and Technology |
| OMI | Open MUMPS Interconnect |
| PSE | Programming Support Environment |
| RDBMS | Relational Database Management System |
| SQL | Structured Query Language |
| VA | Department of Veterans Affairs |

APPENDIX A.2: REFERENCES

A.2.1 MUMPS Language Standards and Specifications

ANSI/MDC X11.1-1984 MUMPS Language Standard. MUMPS Development Committee. 1984.

ANSI/MDC X11.1-1990 MUMPS Language Standard. MUMPS Development Committee. 1990.

"What's in the Next Standard." Thomas Salander, unpublished manuscript. October 1992.

A.2.2 Technical References

Dictionary for Information Systems. American National Standards Institute Standard X3.172-1990. 1990.

"Glossary of Software Engineering Terminology." IEEE Standard 610.12-1990. *IEEE Software Engineering Standards Collection.* Spring 1991.

Application Portability Profile: The U.S. Government's Open System Environment Profile - OSE/I Version. National Institute of Standards and Technology. Draft, 27 October 1992.

The Complete MUMPS: An Introduction and Reference Manual for the MUMPS Programming Language. John Lewkowicz, Prentice-Hall. 1989.

ABCs of MUMPS: An Introduction for Novice and Intermediate Programmers. Richard F. Walters, Digital Press. 1989.

"System Performance: A Benchmark Study of MUMPS and Other Systems." C. M. Alonzo. *MUG Quarterly*: Volume 18, Number 3/4. 1988.

"An Analysis of the Impact of the Argumentless DO on the MUMPS Programming Language." B. Isch. *MUMPS Computing*: Volume 22, Number 3. 1992.

Selection and Use of General Purpose Programming Languages - Overview. U.S. Department of Commerce, NBS Special Publication 500-117 Volume 1. October 1984.

"MUMPS." William J. Harvey and Frederick G. Kohun. Unpublished draft 8 for *Encyclopedia of Microcomputers*. 1991.

"MUMPS: Characteristics and Comparisons with Other Programming Systems." T. Munnecke, R.F. Walters, J. Bowie, C.B. Lazarus, and D.A. Bridger. *Medical Informatics*: Volume 2, Number 3. 1977.

A.2.3 DoD Policy References

"Delegation of Authority and Clarifying Guidance on Waivers from the Use of the Ada Programming Language." Assistant Secretary of Defense: Command, Control, Communications, and Intelligence. 17 April 1992.

"Computer Programming Language Policy." DoD Directive 3405.1. 2 April 1987.